НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр КОВАЛЬ
«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Геометричне моделювання в інформаційних системах»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «Веб-орієнтована система для внесення екологічних параметрів, отриманих з точок збору даних»

Виконав:
студент IV курсу, групи ТР-61
Трофимчук Антон Анатолійович _____

Керівник:
д.т.н., професор
Сліпченко Володимир Георгійович _____

Рецензент:
с.н.с. відділу цифрових моделюючих систем
Інституту проблем реєстрації інформації НАН України,
к.т.н. Сенченко В'ячеслав Родіонович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Геометричне моделювання в інформаційних системах

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр КОВАЛЬ
(підпис)
«___»_____2020р.

ЗАВДАННЯ

на дипломну роботу студенту

Трофимчуку Антону Анатолійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Веб-орієнтована система для внесення екологічних параметрів, отриманих з точок збору даних

керівник роботи д.т.н., професор Сліпченко Володимир Георгійович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від «25» травня 2020р. № **1168-с**

2. Строк подання студентом роботи «17» червня 2020 р.

3. Вихідні дані до роботи мова програмування JavaScript, платформа NodeJS, фреймворк Express, бібліотека React, бібліотеки React-leaflet, react-bootstrap, react ag-grid, каскадна таблиця стилів CSS3, мова розмітки HTML5, система керування базами даних MySQL

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) виконати огляд існуючих рішень, обрати підходящу літературу
2) описати засоби реалізації системи для введення екологічних параметрів
3) розробити програмне забезпечення
4) описати методику роботи користувачів із системою

5. Перелік ілюстративного матеріалу

Мета розробки, аналоги, загальна концепція, схеми роботи системи, структура підсистем, клієнтська частина, серверна частина, база даних, початок роботи із системою, типи об’єктів для внесення, способи внесення інформації, викиди та синхронізація, редагування, довідники, висновки

6. Дата видачі завдання «25» вересня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	01.09.19 – 20.09.19	
2.	Вивчення та аналіз задачі	01.10.19 – 01.11.19	
3.	Розробка архітектури та загальної структури системи	13.10.19 – 25.12.19	
4.	Розробка структур окремих підсистем	15.01.20 – 01.02.20	
5.	Програмна реалізація системи	01.02.20 – 08.02.20	
6.	Оформлення пояснювальної записки	10.02.20 – 15.04.20	
7.	Захист програмного продукту	10.06.2020	
8.	Передзахист	10.06.2020	
9.	Захист	17.06.2020	

Студент _____ _____Трофимчук А.А._____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____Сліпченко В.Г_____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломну роботу виконано на 70 аркушах. Вона містить 3 додатки та перелік посилань на використані джерела з 15 найменувань. У роботі наведено 62 рисунки та 4 таблиці.

Метою даної дипломної роботи є створення веб-орієнтованої системи для внесення екологічних параметрів, отриманих із точок збору даних. Вибір методів для функціонування системи базувався на забезпеченні комфортного способу внесення таких показників. Для демонстрації роботи потрібно створити відповідне програмне забезпечення.

Було розглянуто декілька існуючих аналогів та зроблено порівняння серед них. Основним методом для роботи із системою було обрано метод взаємодії із картою.

Додатково було додано функціонал для роботи із довідниками.

У процесі роботи було побудовано програмне забезпечення, яке демонструє роботу поставленої задачі.

Ключові слова: екологічний стан, екологічні параметри, населення, аналіз, викиди, якість життя, введення показників.

ABSTRACT

Thesis was completed on 70 sheets. It contains 3 appendices and a list of references to used sources from 15 names. The paper contains 62 figures and 4 tables.

The purpose of this thesis is to create a web-based system for entering environmental parameters obtained from data collection points. The choice of methods for the functioning of the system was based on providing a comfortable way to enter such indicators. Appropriate software must be created to demonstrate the work of the system.

Several existing analogues were considered and a comparison was made between them. The method of interaction with the map was chosen as the main method for working with the system. Additionally, functionality for working with dictionaries has been added.

During working process, software was built that demonstrates the work of the task.

Key words: ecological condition, ecological parameters, population, analysis, emissions, quality of life, indicators.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС	Операційна система
Фреймворк	Інфраструктура програмних рішень, що полегшує розробку складних систем.
JS	Мова програмування JavaScript
DOM	Об'єктна модель документа (англ. Document Object Model, DOM) — специфікація прикладного програмного інтерфейсу для роботи зі структурованими документами
HTML	Мова розмітки гіпертекстових документів (англ. HyperText Markup Language) — стандартна мова розмітки веб-сторінок в Інтернеті.
SPA	(англ. Single Page Application) — одно сторінкова веб-сторінка що динамічно маніпулює DOM для відображення різних блоків інтерфейсу
NPM	(англ. Node Package Manager) — менеджер пакетів
React.js	Веб-бібліотека на мові JavaScript
API	(англ. Application Programming Interface) — програмний інтерфейс додатку
URL	(англ. Uniform Resource Locator) — уніфікований вказівник ресурсу
ГДК	Гранично допустима концентрація
ПЗ	Програмне забезпечення

ВСТУП

Питання навколишнього середовища завжди відігравало велику роль у житті суспільства, оскільки люди постійно приділяють увагу умовам життя. У період стрімкого розвитку інформаційних технологій та урбанізації, екологічному стану навколишнього середовища приділяється все більше і більше уваги. У зв'язку зі зростанням споживчого запиту будуються нові об'єкти для виробництва, обробки та видобутку матеріалів, що впливають на навколишнє середовище. Задля забезпечення високого рівня життя та здоров'я існує необхідність створення засобів для зручного слідкування за екологічним станом довкілля.

За допомогою інформаційних технологій проектуються спеціальні комп'ютерні системи з метою отримання точних та актуальних даних у будь-якій точці планети. Такі системи спрощують внесення показників, оскільки усувають потребу пересилання результатів аналізів через пункти передачі. Враховуючи те, що для передачі даних можна використовувати мережу Інтернет, інформацію відразу можна публікувати на відповідних ресурсах, що значно спрощує комунікацію між різними підрозділами моніторингу.

Результатами роботи таких систем є різні види карт, діаграм, таблиць і графіків, що формуються за допомогою різноманітних критеріїв аналізу. Це дозволяє більш точно оцінювати стан та приймати точні рішення задля запобігання можливих екологічних проблем. Проте ці системи розв'язують задачі конкретної області, тому існує потреба у створенні комплексної системи, що дозволить фахівцям із різних сфер бачити повну картину стану навколишнього середовища.

Завданням даної бакалаврської роботи була розробка веб-системи для комфортного внесення екологічних параметрів, отриманих із точок збору даних. Система повинна надавати можливості для додавання, редагування об'єктів та областей, а також забезпечувати роботу із довідниками.

Окремо необхідно було приділити увагу можливості використання системи за допомогою мобільних пристроїв, оскільки ключовою особливістю цієї системи є

орієнтованість на мобільність та апаратну незалежність. Враховуючи це, спеціалістам не потрібно використовувати спеціальну техніку та пристрої з інстальованим програмним забезпеченням, а мати лише доступ до мережі Інтернет.

Оскільки введені показники є важливим елементом екологічного моніторингу, то вони заносяться до віддаленої бази даних. Таким чином, фахівці з інших регіонів та галузей мають змогу відразу бачити повну картину та приймати необхідні рішення.

Дана веб-система може бути використана фахівцями з галузей екологічного та економічного моніторингу, підприємств або фірм, що пов’язані із забезпеченням високого рівня життя та належного стану навколишнього середовища.

1. ЗАДАЧА СТВОРЕННЯ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ ДЛЯ ВНЕСЕННЯ ЕКОЛОГІЧНИХ ПАРАМЕТРІВ

Метою даної дипломної є створення веб-орієнтованої системи для внесення екологічних параметрів, отриманих з точок збору даних.

Для розв’язання поставленої задачі необхідно застосувати метод роботи з картою, що буде використовуватись для внесення екологічних параметрів та розробити програмне забезпечення для реалізації обраного методу.

Призначення системи полягає у:

— мобільності;

— розширенні можливості одночасного внесення даних різними експертами з різних точок незалежно від програмного та апаратного забезпечення.

Оскільки система є орієнтованою на веб, вона має бути доступна за наявності Інтернету та не залежати від операційної системи.

Програмний продукт повинен мати наступний функціонал:

— можливість наносити маркери на карту;

— можливість наносити полігони на карту;

— можливість редагувати існуючі об’єкти;

— можливість вносити інформацію про викиди речовин залежно від обраного середовища;

— можливість користувачів шукати інформацію у довідниках;

— можливість адміністратора вносити нову інформацію у довідники;

— можливість адміністратора редагувати інформацію у довідниках;

— можливість адміністратора видаляти інформацію із довідників.

Після реалізації програмного забезпечення потрібно протестувати його на правильність виконання та провести аналіз отриманих результатів.

2. АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ПОСТАВЛЕНОЇ ЗАДАЧІ

Сьогодні розроблені різноманітні інформаційні системи, що призначені для моніторингу навколишнього середовища. Головними їх задачами є оцінка антропогенного впливу, проведення досліджень щодо стану біосфери, прогнозування та оцінка змін для виявлення негативних природних процесів. Оскільки задачі моніторингу доволі різнобічні, тому зазвичай їх класифікують на різні області моніторингу [1]. Кожна із них виконує чітко поставлену задачу спостереження за конкретною областю біосфери.

По критеріях зазвичай виділяють такі типи моніторингу:

— кліматичний — прогноз та оцінка перемін клімату, а також моніторинг стану кліматичної системи;

— літомоніторинг — оцінка стану геологічного середовища та визначення впливу людини на його стан;

— біосферний — зміни у біосфері, включаючи антропогенний вплив;

— біоекологічний — метою є захист здоров’я людини від впливу шкідливих чинників навколишнього середовища;

— геофізичний — прогноз та моніторинг забруднення середовища;

— геоекологічний — слідкування за перемінами природних екологічних систем для прогнозування стихійних лих;

— біологічний — моніторинг навколишнього середовища, що базується на спостереженні за живими організмами;

— супутниковий моніторинг — моніторинг, для якого використовують аеро- та космофотозйомку;

Технологічний прогрес спонукає до виникнення спеціальних комп’ютерних системи, що спрощують спостереження за станом довкілля. У таблиці 2.1 наводяться приклади різних систем класифікованих за областями екомоніторингу, що дозволяють провести аналіз того, як наявні наразі рішення покривають різні області моніторингу.

Таблиця 2.1. Комп’ютерні системи екомоніторингу

Система	Біосферний моніторинг	Біоекологічний моніторинг	Геоекологічний моніторинг	Літомоніторинг	Геофізичний моніторинг	Кліматичний моніторинг	Біологічний моніторинг	Спутниковий моніторинг
СЕМОС	+	+	+	+	+	+		
ЕПК РОСА	+				+			
КС «МЕРАS»		+					+	
ГСМ-«ВІДКРИТЕ ДОВКІЛЛЯ»			+		+			
ПЗ «НОСТРАДАМУС».				+			+	
ІНВЕНТЕР					+			
КС «RECASS»						+	+	
АСРРМ							+	+
ПЗ «SULTAN»							+	
ПЗ «ДОЗА»							+	
NEORIST								+
ЕРА-Повітря			+	+				
СМСД-Донецької області	+	+						

Основними функціями таких систем є:

— розрахунок викидів шкідливих речовин на виробництвах;

— інвентаризація викидів на заводах і підприємствах по забруднювальних речовинах;

— розрахунки концентрацій шкідливих речовин;

— розрахунки приземної концентрації забруднювальних речовин, що містяться у викидах газокомпресорних станцій;

— моделювання з використанням геоінформаційних систем — графічне зображення об’єктів дослідження систем екомоніторингу за допомогою карт, аеро- та космофотозйомок [2].

Екологічний програмний комплекс «ЕПК РОСА» — комплекс для моделювання у сфері промислової екології. Програма спрямована на розробку документації промислових організацій із фокусом на екологічні показники. Система дозволяє використовувати екологічні ГІС, що забезпечують можливість використання просканованих зображень підприємства як карту. Має зручний інтерфейс та великі можливості для складних обчислень. Також, є спеціальна демоверсія для ознайомлення, що дозволяє спробувати основні функції даного екологічного програмного комплексу. Просторова інформація вводиться на цифровій карті за допомогою інструментів графічного модуля «ЕкоГІС».

Введення даних у цій системі відбувається за допомогою джерел картографічної інформації. Для ЕПК РОСА можливе використання наступних джерел:

— растрові зображення, що можуть бути у форматах JPEG, JPG, BMP;

— цифрові карти;

Схема процесу внесення параметрів зображена на рисунку 2.1 [3].

Рисунок 2.1 — Схема процесу роботи системи ЕПК РОСА

Для внесення екологічних параметрів об’єктів використовується формат роботи із таблицею, куди вносяться відповідні дані. У таблиці визначені усі параметри, що необхідні для розрахунків. Після цього, вибираються фігури, що накладаються на карту окремим шаром. На рисунках 2.2 та 2.3 продемонстровано роботу з даною системою [3].

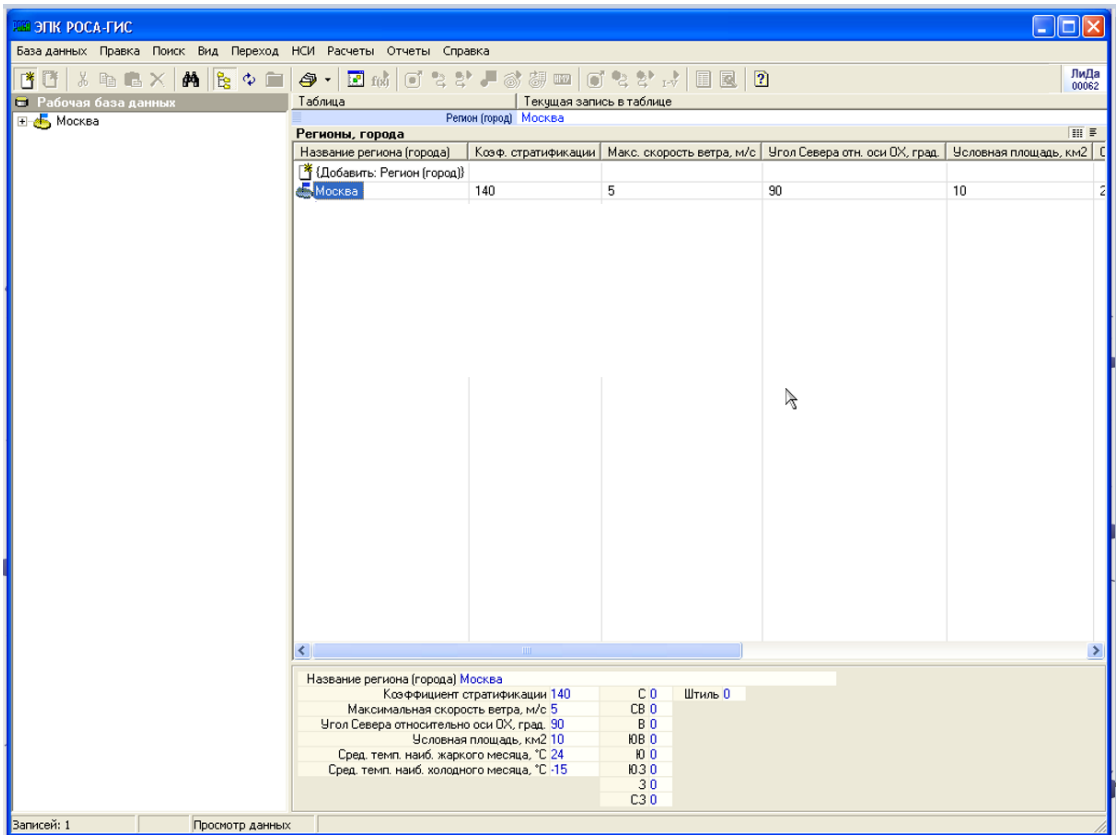


Рисунок 2.2 — Введення екологічних параметрів

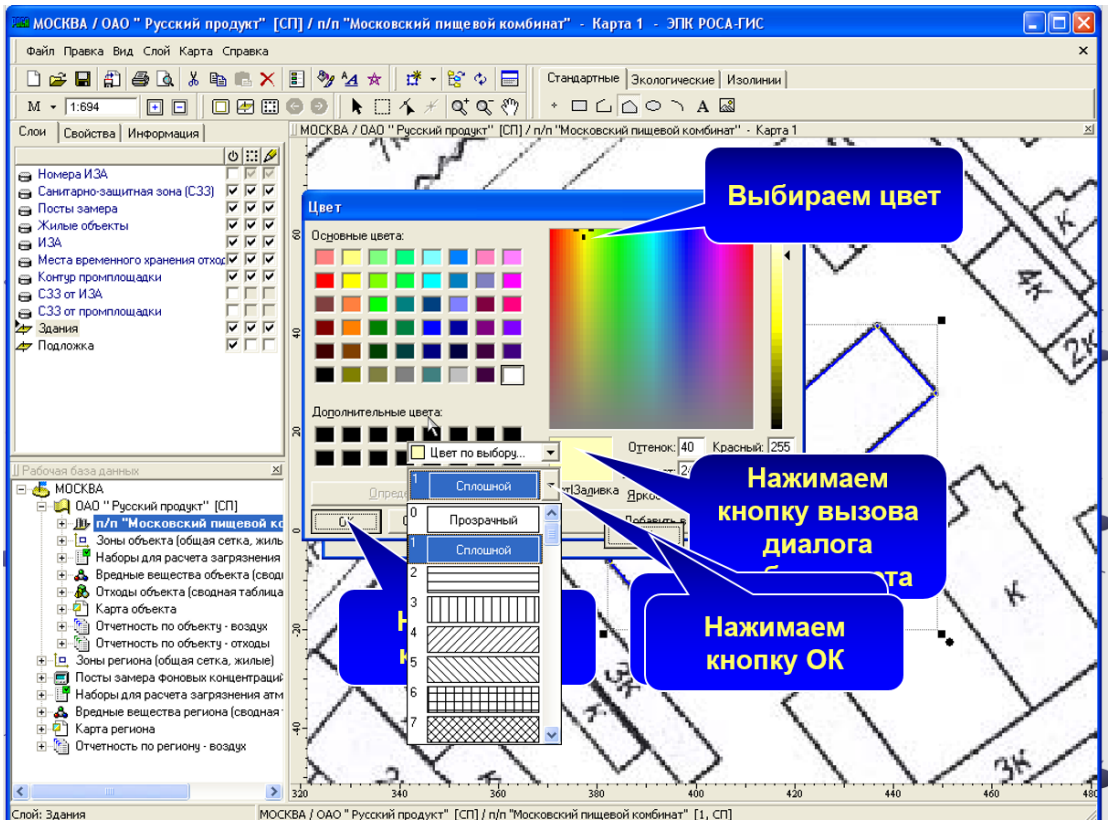


Рисунок 2.3 — Опції нанесення області на карту

Недоліком даного ПЗ є система купівлі та інсталяції конфігурацій. Кожна конфігурація для певного розділу купляється окремо, як окремий блок функціоналу, що накладає значні обмеження на можливості внесення даних у систему. Наприклад, якщо користувач придбав конфігурацію системи по розділу «Повітря» і виникла необхідність внесення параметрів по розділу «Відходи», то він повинен купити окремо відповідний набір програмного забезпечення та встановити його. Таким способом, користувач отримує доступ до завантаження та інсталяції блоку програмного забезпечення ЕПК РОСА.

ПЗ «НОСТРАДАМУС» — комп'ютерна програма, призначення якої полягає у прогнозуванні радіоактивного опромінення від викидів радіоактивних речовин при аваріях на атомних електростанціях та інших підприємствах, пов'язаних із радіоактивними речовинами. Система використовується для прийняття рішень в реальному часі у момент початкової фази радіаційної аварії. Дані, які може розраховувати дана система:

— концентрації радіоактивних речовин;

— приземні концентрації для кожного радіонукліда;

— величину дози від кожного із радіонуклідів (або суму від усіх) на різні органи людини, із включенням до розрахунку параметрів різних вікових груп та шляхів опромінення.

— розрахунок зовнішнього опромінення від радіоактивної хмари, а також від забруднених поверхонь;

Важливою рисою системи є можливість введення джерел довільної форми (точкові, плоскі, об'ємні) й конфігурації, зі змінними у часі параметрами, та обчислення розповсюдження радіонуклідів на відстані у десятки або сотні кілометрів. При розрахунках враховуються такі фактори, як рельєф місцевості, неоднорідність і тимчасові зміни вітрового поля, опади в зоні поширення радіоактивного сліду.

Введення даних відбувається шляхом внесення параметрів по різних категоріях, що дозволяють отримувати більш точні результати обчислень доз радіоактивного опромінення. Приклад задання параметрів зображено на рисунку 2.4 [4].

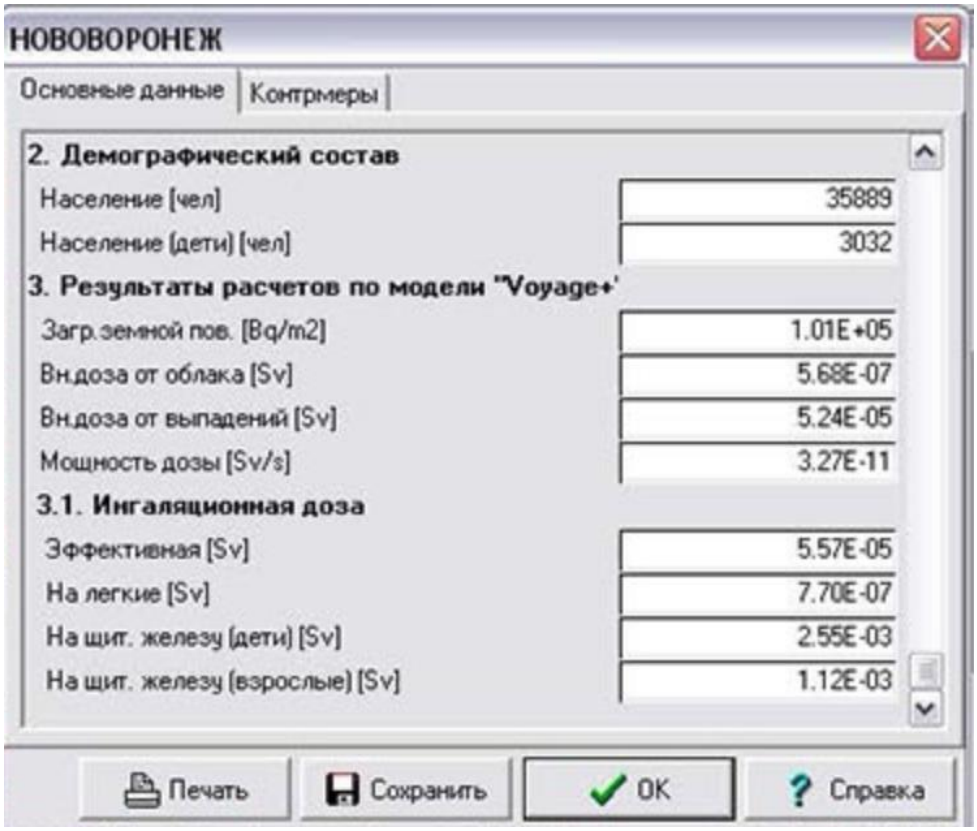


Рисунок 2.4 — Приклад введення даних у ПЗ «НОСТРАДАМУС»

Оскільки система проводить складні розрахунки, що базуються на вхідних параметрах радіоактивності, то існують певні вимоги до системної конфігурації машини, де будуть проводитись обчислення. Якщо конфігурація комп’ютера нижче від рекомендованої, система буде працювати повільно та з можливими помилками у роботі, накладаючи труднощі для внесення нових показників.

RECASS NT — система, що була розроблена науково-виробничим об’єднанням «Тайфун» у 1993 році. Завданням системи є допомога при прийнятті рішень у ситуаціях виникнення радіаційних аварій. Застосовується для швидкого аналізу та обробки даних радіоактивного забруднення населених пунктів. Система призначена для моделювання на операційній системі Microsoft Windows. Серверна частина системи працює під управлінням Microsoft Windows Server 2012 або вище, а клієнтська на операційних системах Microsoft Windows 7 або вище. Програмне забезпечення даної системи дозволяє здійснювати:

— обчислення дозованого навантаження;

— можливість одночасного доступу до розрахунків та оперативної інформації;

— обробка оперативних даних щодо радіаційної ситуації;

— фундамент для створення рекомендованих дій та засобів при радіаційних аваріях.

Основною особливістю цієї системи є обновлюваний банк прогнозованих та актуальних метеорологічних даних по всій земній поверхні, що дозволяє оперативно підготовлювати прогнози у випадку аварій на будь-якому об’єкті [5]. Демонстраційний рисунок 2.5 було запозичено з офіційної документації [6].

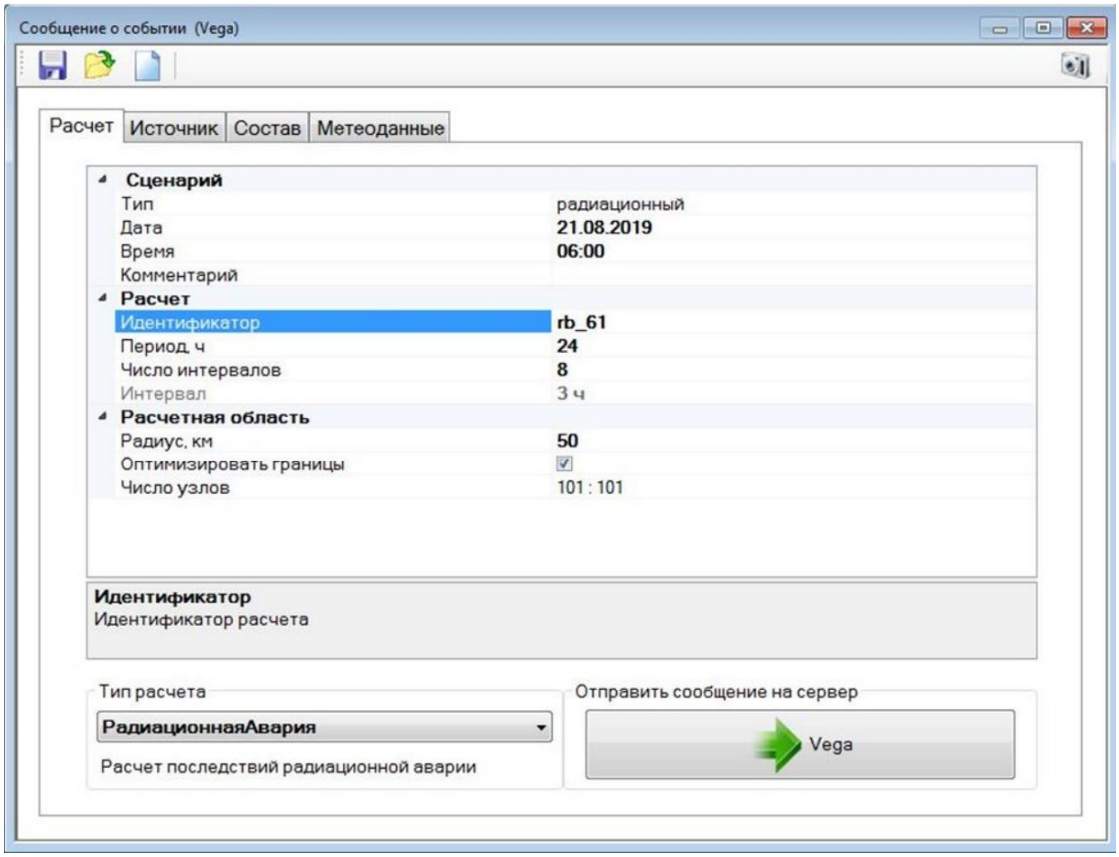


Рисунок 2.5 — Задання сценарію у системі RECASS NT

Введення до системи параметрів в основному складається із такої послідовності дій:

— запуск спеціального вікна «Термінала»;

— формування сповіщення про аварію;

— завантаження спеціального файлу сценарію аварії та можливе його коригування;

— відправлення повідомлення на сервер;

— отримання і перегляд результату;

Як видно вище, недоліком даної системи є прив’язка вхідних даних до спеціального сценарію, що може задавати певні обмеження для користувача системи. Також спрямованість даного програмного забезпечення на операційну систему Windows, з додатково встановленим середовищем «Microsoft .NET Framework 4.0», накладає значні обмеження на можливості використання даного програмного забезпечення на інших операційних системах.

2.1 Висновки до розділу

Після огляду аналогічних систем, було виявлено, що основними недоліками таких систем є необхідність інсталяції та системні вимоги до пристрою, який повинен забезпечувати правильне функціонування програмного забезпечення. Враховуючи це, для внесення параметрів фахівець повинен мати доступ до комп’ютера та спеціального програмного забезпечення. Веб-орієнтовані системи позбавлені таких недоліків, що робить можливим внесення параметрів з будь-якого пристрою та без інсталяції ПЗ. Таким чином, веб-системи є актуальним способом забезпечення швидкого доступу для внесення екологічних параметрів.

3. ЗАСОБИ РЕАЛІЗАЦІЇ СИСТЕМИ

Для забезпечення вимог до програмного продукту даної системи, було використано інструменти, які зображені на рисунку 3.1.

Рисунок 3.1 — Інструменти розробки

Для реалізації було використано:

— середовища розробки WebStorm IDEA та Visual Studio Code;

— Git для задання версій розроблюваної системи;

— мову гіпертекстової розмітки HTML;

— каскадну таблицю стилів CSS з використанням бібліотеки react-bootstrap для задання уніфікованої стилістики системи;

— реляційну базу даних MySQL;

— бібліотеку React для оформлення клієнтської частини;

— мову програмування JavaScript для клієнтської та серверної частини;

— розширення JSX для більш гнучкого описання компонентів React;

— платформу NodeJS для серверної частини;

— фреймворк Express для серверної частини;

— бібліотеку React Ag-Grid для роботи із довідниками.

Оскільки дана система орієнтована на веб, тому було розглянуті варіанти, що забезпечують можливість користування незалежно від пристрою. Такими обрано незалежні від ОС технології — NodeJS, HTML, CSS, React.

3.1 Мова програмування JavaScript

Основною мовою програмування була обрана мова програмування високого рівня JavaScript, що забезпечує кросплатформність, малий обсяг продакшин-коду, а також через платформу NodeJS з менеджером пакетів NPM, який дозволяє легко керувати залежностями та додавати до застосунку лише ті пакети, які необхідні проекту [7]. Це дозволяє легко масштабувати й оновлювати програмний продукт.

Оскільки мова JavaScript підтримується будь яким браузером, це дозволяє писати системи, не використовуючи трансляторів, що перетворюють код програми. Використовуючи цикл подій JavaScript доволі гнучко дозволяє працювати з асинхронними задачами, що забезпечує високу швидкодію програми. Загальна схема роботи циклу подій зображена на рисунку 3.1.1. Використовуючи стек задач, WebAPI та чергу вдається досягти передбачуваного стану на кожному етапі виконання програми.

Мова JavaScript із самого початку розроблялась для виконання скриптів на стороні клієнта, щоб додати деяку «динаміку» сторінкам. Тому JavaScript використовує так звану модель DOM (англ. Document Object Model) для взаємодії із документом. Ця модель дозволяє взаємодіяти з елементами сторінки і навішувати обробники подій на потрібні елементи. Наприклад, коли користувач натискає купити товар, спрацьовує обробник події, що перенаправляє користувача на сторінку для оформлення замовлення. За допомогою цієї моделі можна програмно видаляти, переставляти, додавати цілі блоки HTML, що дозволяє забезпечити високу гнучкість сторінки та зробити її «живою».

Додатково JavaScript слідкує за розподілом пам’яті в програмі та вчасно очищає

непотрібні дані. За це відповідає збирач сміття (англ. garbage collector), що звільняє пам'ять у момент, коли об’єкти втрачають свою актуальність. Важливим пунктом розподілу пам’яті є те, що JavaScript виділяє її динамічно, тільки за необхідності. Через це вдається позбутися типових помилок, як-от вихід за межі масиву чи операції з неіснуючими комірками пам’яті.

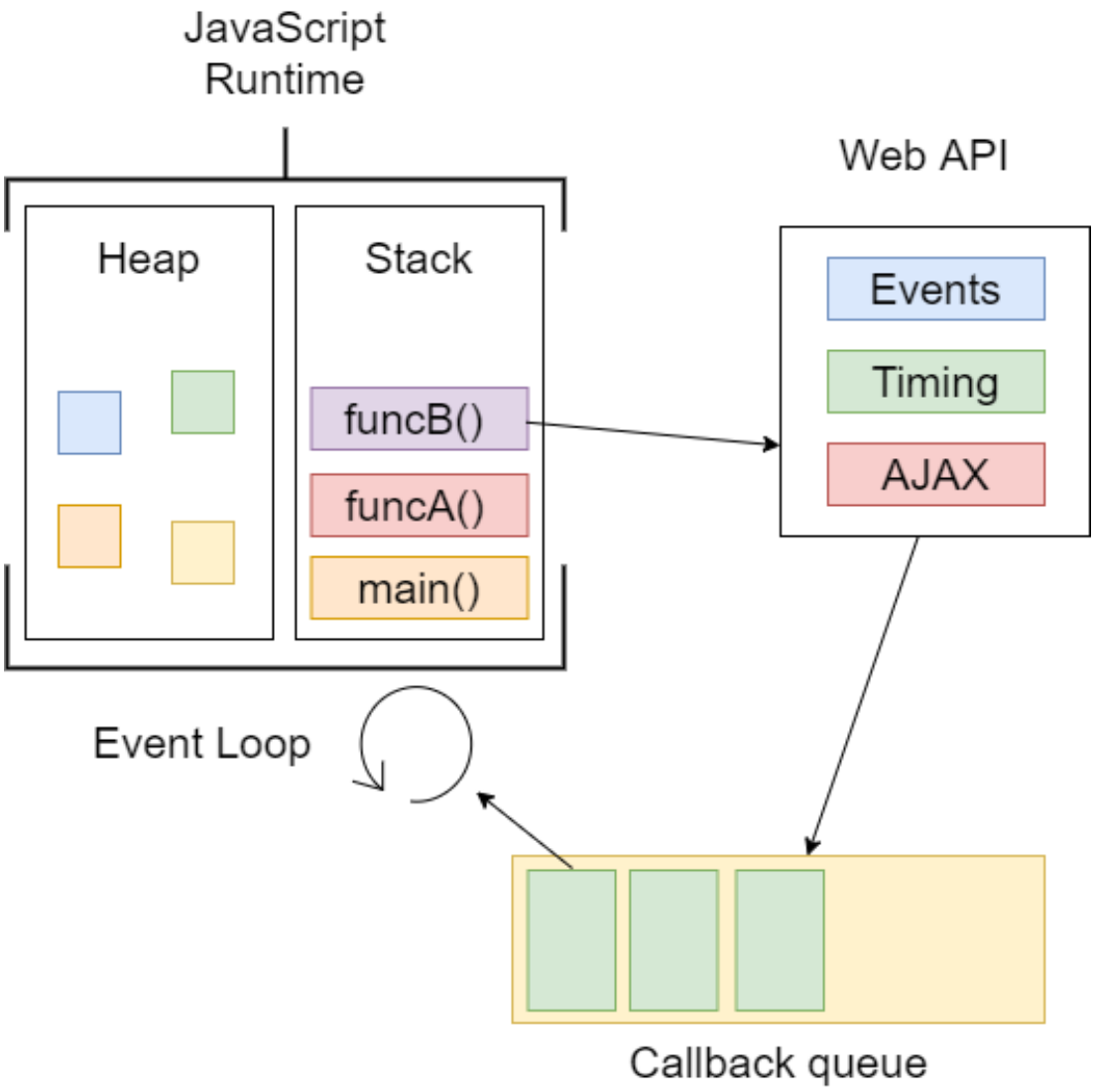


Рисунок 3.1.1 — Схема роботи циклу подій у JavaScript

Декілька слів хочеться додати про платформу NodeJS, оскільки вся серверна частина даної веб-системи побудована на ній. NodeJS — це серверна платформа для роботи із мовою програмування JavaScript на базі «двигуна» V8. V8 — це «двигун» із відкритим кодом, написаним мовою програмування C++. Він використовує JavaScript

код і перетворює його у швидший машинний, який комп’ютер може запускати без потреби попередньої інтерпретації. NodeJS дозволяє працювати зі сторонніми бібліотеками та пакетами, що можуть бути завантажені через пакетний менеджер NPM (англ. Node Package Manager). Перевагою NodeJS є те, що з ним легко масштабуватись. У випадку, коли на сервер намагаються зайти тисячі користувачів, дана платформа працює асинхронно, тобто розраховує пріоритети та виділяє ресурси більш грамотно. У такому ж випадку, Java, наприклад, виділяє на кожне з’єднання окремий потік. Для забезпечення асинхронної роботи також використовується механізм циклу подій, як було зазначено раніше.

Таким чином мова програмування JavaScript є хорошим вибором для створення веб-систем, включаючи можливість взаємодії із великою кількістю складних інтерфейсів.

3.2 Мова розмітки HTML

Для показу результатів у вікні браузера користувача використовується мова розмітки HTML (англ. HyperText Markup Language), оскільки вона є основним засобом зображення контенту веб-сторінок [8]. Браузер показує сторінку користувачу як результат обробки HTML документа. Разом із каскадною таблицею стилів CSS та скриптами формуються сторінки, які ми звикли бачити у браузері.

Актуальна мова розмітки HTML5 дозволяє:

— розробку семантично структурованого HTML документа шляхом виділення таких змістовних блоків, як хедери, основні частини, списки, секції, параграфи, заголовки, таблиці;

— створення форм;

— використання внутрішніх сховищ браузера для зберігання даних;

— додавання зображень, відео та інших видів медіаконтенту;

— використання веб-сокетів;

— використання тегу canvas для малювання і роботи із графікою;

Важливо зазначити, що в HTML виділяють поняття «валідності». Для отримання «валідного» HTML необхідно дотримуватись спеціально визначених правил атрибутів та контенту елементу. Повне дотримання цих правил забезпечує більшу доступність HTML документів, а також це покращує результати рейтингу цієї сторінки в мережі Інтернет. Для редагування HTML можна користуватись будь-яким текстовим редактором, а для перегляду документу використовується браузер.

3.3 Каскадна таблиця стилів CSS

Оскільки систему для введення параметрів необхідно було візуально оформити, була використана каскадна таблиця стилів CSS (англ. Cascading style sheets). CSS — це мова, що використовується для описання стилів для HTML документа [9]. Це дозволяє описувати стиль HTML документу, як він має бути оформлений та показаний. Каскадна таблиця використовується для задання кольору, шрифтів, полів, висоти та ширини, позиціювання елементів та іншої візуальної стилістики.

Плюсами зв’язку HTML і CSS є те, що вони виконують різні задачі. HTML відповідає за сам зміст веб-сторінки, а CSS — за вигляд і візуальну складову. Це дозволяє один і той самий HTML документ оформити різними таблицями стилів, що в результаті дає дві оформлені сторінки та покращує гнучкість всієї системи. Перевагами каскадної таблиці стилів є:

— можливість стилізувати один і той же елемент у залежності від умов. Наприклад, змінювати стилі при наведенні чи при зменшенні розміру вікна;

— прискорення завантаження веб-сторінок через розподілення окремо стилів та окремо розмітки;

— розділення .css файлів у залежності від смислового навантаження;

— точний контроль над зовнішнім виглядом веб-сторінок;

Для забезпечення однакової стилістики веб-системи було використано

бібліотеку стилів react-bootstrap, яка дає можливість використовувати стилізовані елементи та css класи [10].

3.4 Бібліотека React

У зв’язку зі стрімким розвитком технологій, все більше уваги приділяється інструментам для розробки. Ці інструменти існують для того, щоб полегшити сам процес розробки та дати можливість легко масштабувати й підтримувати програмне забезпечення. Таким інструментом було обрано бібліотеку React, яка забезпечує хорошу швидкодію та відносну простоту у розробці [11]. Робота з цією бібліотекою являє собою написання компонентів, що, взаємодіючи із собою, створюють складні інтерфейси для роботи користувача.

Кожен компонент має свою чітко визначену роль і виконує поставлену перед ним задачу. Інкапсуляція логіки всередині компоненти дозволяє полегшити можливість перевикористання. Компоненти можуть приймати вхідні параметри, а потім обробляти їх згідно з поставленою задачею. Це дає змогу не вдаватись у деталі реалізації компонентів, а лише забезпечити набір вхідних параметрів («контракт») для них і всі вхідні дані будуть опрацьовані належним чином.

У основі створення інтерактивних інтерфейсів React використовує декларативний спосіб опису. Розробнику необхідно лише описати, як різні частини інтерфейсу виглядають у кожному стані і React ефективно оновить й перемалює лише потрібні компоненти, коли вхідні дані зміняться [12]. Для забезпечення такої роботи застосунків, React використовує спеціальну абстракцію VirtualDOM (рисунок. 3.4.1). Декларативні інтерфейси роблять код більш передбачуваним і його значно легше налагоджувати.

Ефективним «синтаксичним цукром» є розширення JSX для React, що дозволяє використовувати JavaScript прямо у шаблонах. Це значно спрощує взаємодію із DOM та спосіб показу динамічних даних, тому що немає потреби напряму звертатись до

DOM і вручну міняти їх. Все це робить JSX, що дозволяє уникнути важко відловлюваних помилок, непередбачуваної поведінки системи та зменшити загальну кількість коду.

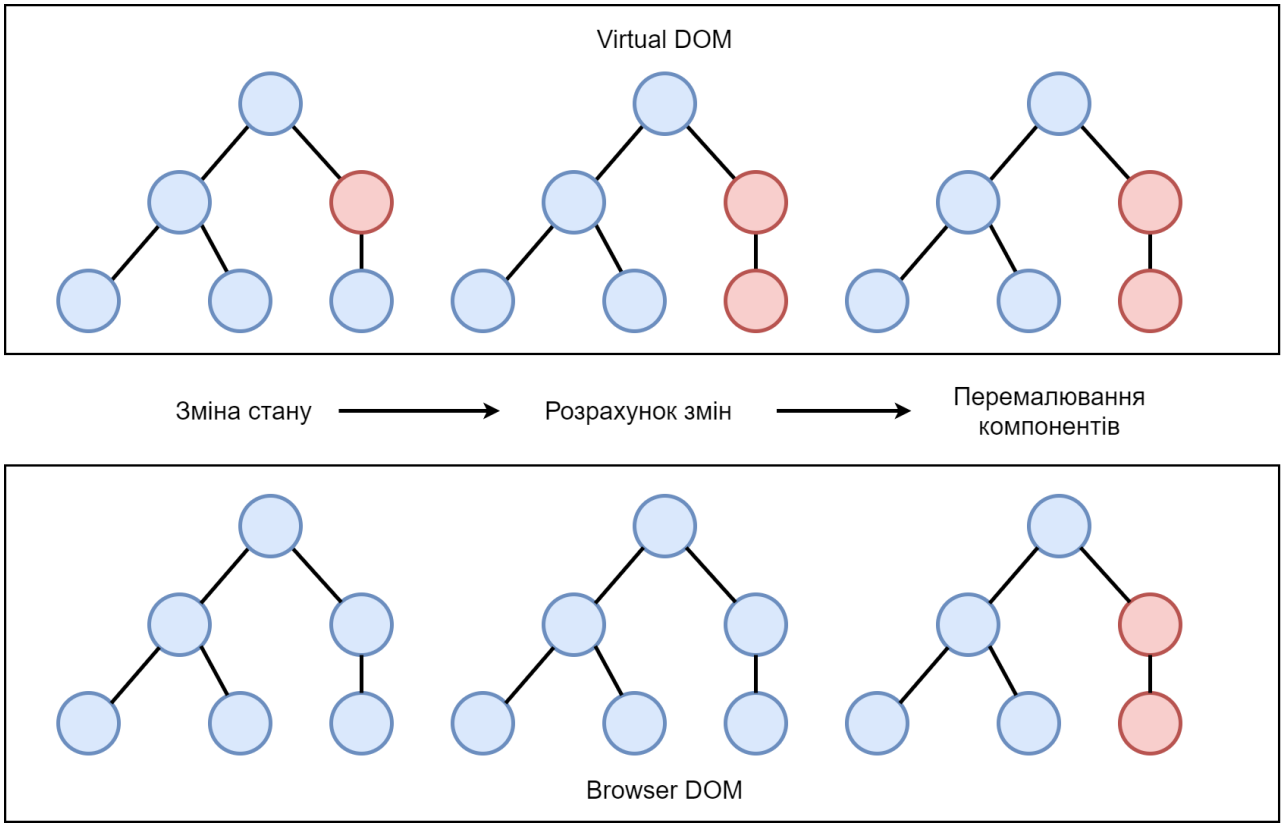


Рисунок 3.4.1 — Перемалювання тільки «потрібних» компонент за допомогою VirtualDOM

Додатково для React є офіційна утиліта Create React App, що дозволяє швидко розгортати середовище для розробки. Дана утиліта була застосована у розроблюваній веб-системі, що дозволило більш точно сконцентруватись на розробці функціоналу. Під капотом у цієї утиліти є статичний збірник модулів, який дозволяє правильно і найбільш оптимізовано збирати всю систему. Додатково були введені команди, що призначені для підготовки системи до розробки, а також для збірки production-build, що містить максимальну мініфікацію та обфускацію коду, дозволяючи отримати мінімальний розмір фінальної збірки.

3.5 Система контролю версій Git

Для задання версій програмного продукту було обрано систему Git, як найбільш популярну та перевірену часом. До виникнення систем контролю версій, розробка програмного забезпечення часто стикалась із труднощами ведення версій програмного продукту. Часто виникали проблеми вирішення актуального стану проекту. Іноді невеликі зміни у програмному коді спричиняли повний вихід системи із ладу й пошук таких змін був надзвичайно складною задачею. Відомі методи ведення версій, що прив'язувались до дати зміни файлу, а також методи індексів, коли до проекту після кожної зміни додавався новий індекс. Проте усе це не вирішувало основної проблеми підтримки та ведення версій проекту. Враховуючи такі проблеми, почались розробки спеціально направленого програмного забезпечення, що дозволяє зручно слідкувати за кожною зміною у проекті. Такою системою став Git.

Суть роботи Git у тому, щоб робити «знімки» стану проекту на момент коміту (англ. commit). Коміт — це об'єкт, який містить посилання на знімок стану проекту, метадані автора, коментарі та вказівники на батьківські коміти. Коміти можна з'єднувати в один, розбивати на декілька, міняти місцями, міняти коментарі коміта, видаляти коміти, копіювати з однієї гілки в іншу. У такий спосіб, вся розробка складається із множини комітів, кожен із яких можна знайти і визначити які зміни робились, коли вони були зроблені та ким (рисунок 3.5.1). Тому не виникає потреби вручну проставляти індекси та робити будь-яку додаткову роботу.

До плюсів Git можна віднести наявність двох репозиторіїв: локального, що зберігається на комп'ютері розробника і дозволяє працювати навіть без мережі Інтернет та віддаленого, що знаходиться на сервері, де відбувається публікація виконаної роботи. Для даної веб-системи у якості віддаленого репозиторію використовувався Github.

Github надає широкі можливості для формування кодової бази та процесу розробки. Зливання завершеної частини коду у головну вітку зазвичай відбувається за допомогою спеціального «реквесту» (англ. Merge Request), який показує історію

розробки цього функціоналу та всі зроблені зміни. З цим «реквестом» тісно пов'язаний процес перегляду змін іншими розробниками (англ. Code Review), за допомогою якого, розробники обговорюють рішення і залишають свої коментарі щодо покращення коду або уникнення потенційних помилок. Доступ до цього репозиторію здійснюється в основному за допомогою SSH або HTTP з'єднання.

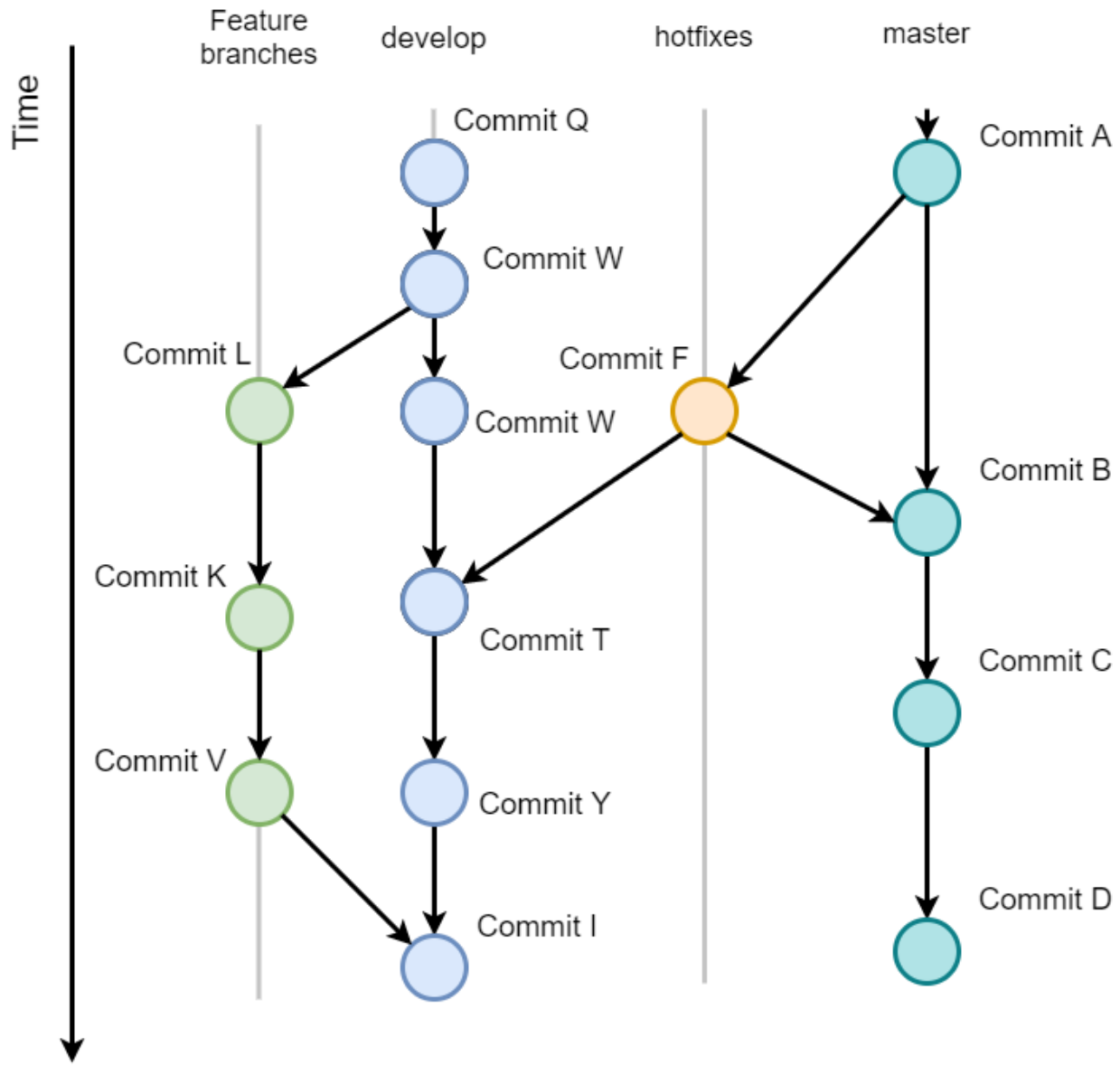


Рисунок 3.5.1 — Приклад процесу розробки за допомогою Git

Під час розробки веб-системи Git дозволяв стежити за змінами й швидко знаходити помилки у програмі. Також, з метою розподілу різних частин функціоналу,

часто використовувались гілки, що забезпечували зручний спосіб для ведення розробки.

3.6 Фреймворк Express

Щоб забезпечити швидкодію із малим фінальним розміром серверної частини було обрано фреймворк Express [13]. Цей фреймворк є гнучким, мінімалістичним та швидким інструментом для роботи серверної частини. Він представляє широкий набір функцій та можливостей для мобільних та веб-застосунків. Маючи у своєму розпорядженні багато службових методів HTTP і проміжних обробників, створити надійний API можна швидко і легко. Express представляє тонкий шар фундаментальних функцій веб-додатків, які не заважають розробнику працювати із давно відомими та перевіреними часом можливостями NodeJS. Враховуючи мінімалістичність фреймворку, масштабування програмного продукту відбувається легко, без великої кількості додаткового коду, що у результаті дає програмну реалізацію, що легко підтримувати у майбутньому.

Окремо варто відзначити спільноту та документацію. Через свою гнучкість та мінімалістичність Express заслужив довіру великої кількості розробників і біля нього сформувалось міцна спільнота. Це дозволяє розробникам легко ділитись своїми напрацюваннями та «кращими методиками» (англ. «best practices»). У випадку виникнення будь яких запитань або труднощів із реалізацією того чи іншого функціоналу, існують багато статей з детальними поясненнями, що майже завжди гарантує розв’язок, оскільки хтось уже стикався зі схожими проблемами. Детальна документація на офіційному сайті забезпечує комфорт при розробці, адже пояснення до інструменту, який використовується, забезпечує надійність розроблюваної архітектури. Документація високо ціниться розробниками, бо надає чітке усвідомлення того, як працює API й що необхідно для реалізації тих чи інших модулів.

3.7 Реляційна база даних MySQL

Однією із найпопулярніших систем управління базами даних є MySQL [14]. База даних являє собою структуровану сукупність даних. Ці дані можуть бути будь-якими — від простого списку до цілих кластерів інформації у корпоративній мережі. Оскільки в реляційній базі даних всі інформація зберігається не купою, а в окремих таблицях, тому досягається значний виграш у швидкості й гнучкості системи. Таблиці пов’язуються за допомогою відношень, що забезпечують можливість об’єднання даних із декількох таблиць. MySQL — це ПЗ з відкритим вихідним кодом, який може використовувати та модифікувати будь-хто. Вважаючи це, було розроблено багато можливостей у тісному контакті з користувачами. Цей інструмент було обрано через свою швидкодію, надійність і легкість у користуванні.

Одним із корисних інструментів MySQL при розробці даного програмного продукту був MySQL Workbench — програма для візуального представлення та проектування баз даних (рисунок 3.7.1).

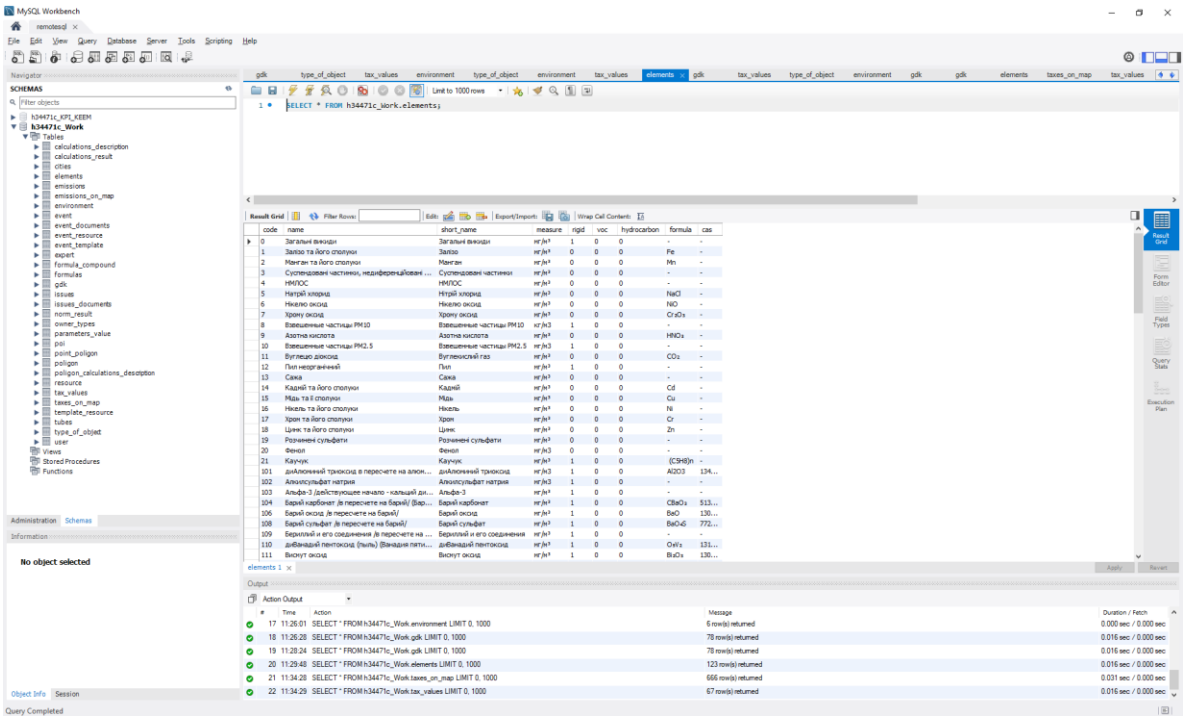


Рисунок 3.7.1 — Вигляд вікна MySQL Workbench

Цей інструмент дозволяє створювати, моделювати, проектувати, редагувати та проводити інші дії із базами даних, не використовуючи інших додаткових засобів. У цьому сервісі представлена можливість використання для баз даних як локального MySQL — сервера, так і віддаленого.

Враховуючи вищесказане, MySQL Workbench часто застосовувався при розробці, оскільки база даних для веб-системи зберігається на віддаленому сервері.

3.8 Бібліотека React Ag-Grid для роботи із таблицями

Дана бібліотека відома через свої широкі можливості для роботи із таблицями на веб-сторінках і серверній стороні. Вона має підтримку API для роботи як зі звичайним JavaScript, так і з усіма основними фреймворками та бібліотеками. Бібліотеку можна легко приєднати до Angular, Vue та React застосунків, тому вона високо оцінюється у колах розробників. Оскільки дана веб-система побудована на базі React бібліотеки, відповідно була використана версія Ag-Grid для React. Приклад роботи бібліотеки зображено на рисунку 3.8.1 [15].

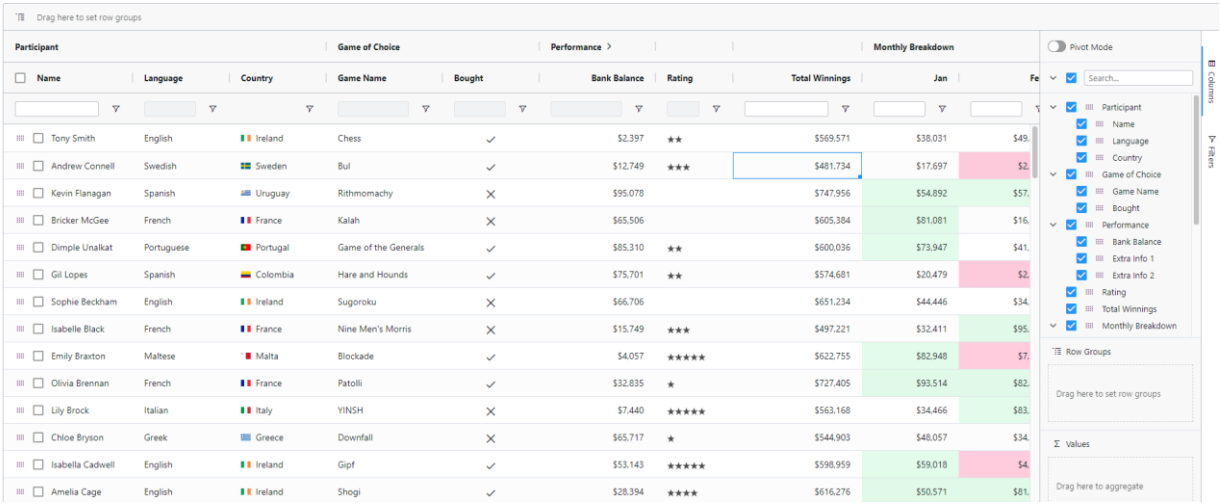


Рисунок 3.8.1 — Зовнішній вигляд таблиці побудованої за допомогою Ag-Grid

Головними перевагами даного інструменту є широкі можливості для налаштування зовнішнього вигляду та стилізації таблиць, можливість роботи як на стороні клієнта, так і на стороні сервера, а також опції для вибірки та фільтрації елементів. За окрему плату пропонуються спеціальні панелі та сайд-бари, інтегровані графіки та компоненти. Для даної веб-системи було використано функціонал із безкоштовної версії бібліотеки, що покриває усі необхідні операції із таблицями.

3.9 Висновки до розділу

Для досягнення ефективної роботи веб-системи було обрано інструменти та технології, що не залежать від операційної системи та забезпечують швидкодію всіх її компонентів. Обрані технології відповідають сучасним вимогам до розробки й дозволяють легко масштабувати та підтримувати програмний продукт.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1. Загальна структура системи

Для реалізації веб-орієнтованого додатку була використана клієнт-серверна архітектура (рисунок 4.1.1).

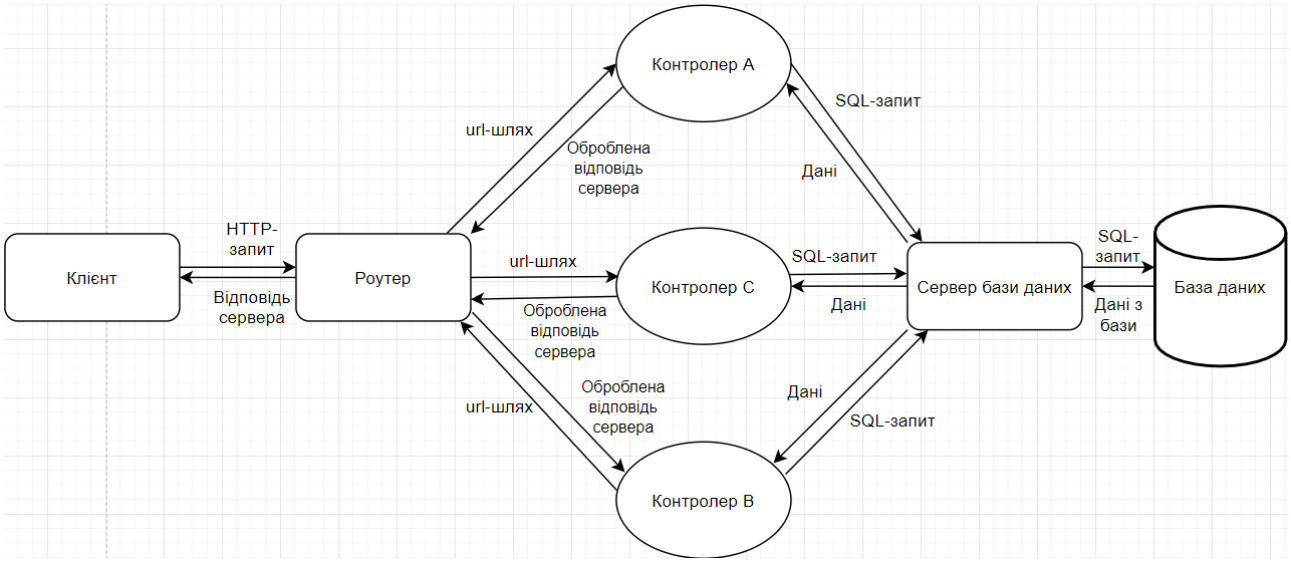


Рисунок 4.1.1 — Схема клієнт-серверного застосунку

Загальна структура даної веб системи складається із клієнтської частини, серверної та сервера бази даних (рисунок 4.1.2).

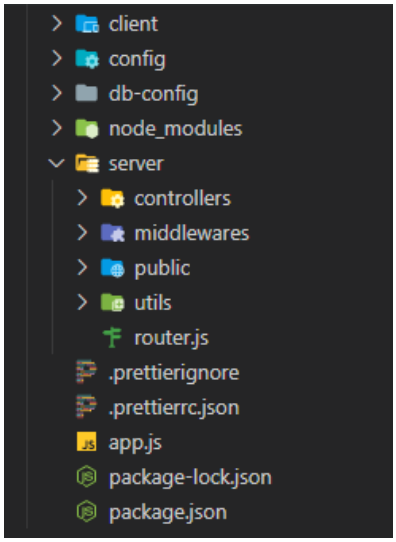


Рисунок 4.1.2 — Структура проекту

Така архітектура дозволяє розгорнути всю систему на будь-якій операційній системі й дозволить працювати із нею використовуючи будь-який браузер.

Підсистеми, що були розроблені у даній дипломній роботі, виділені на рисунку 4.1.3 синім кольором.

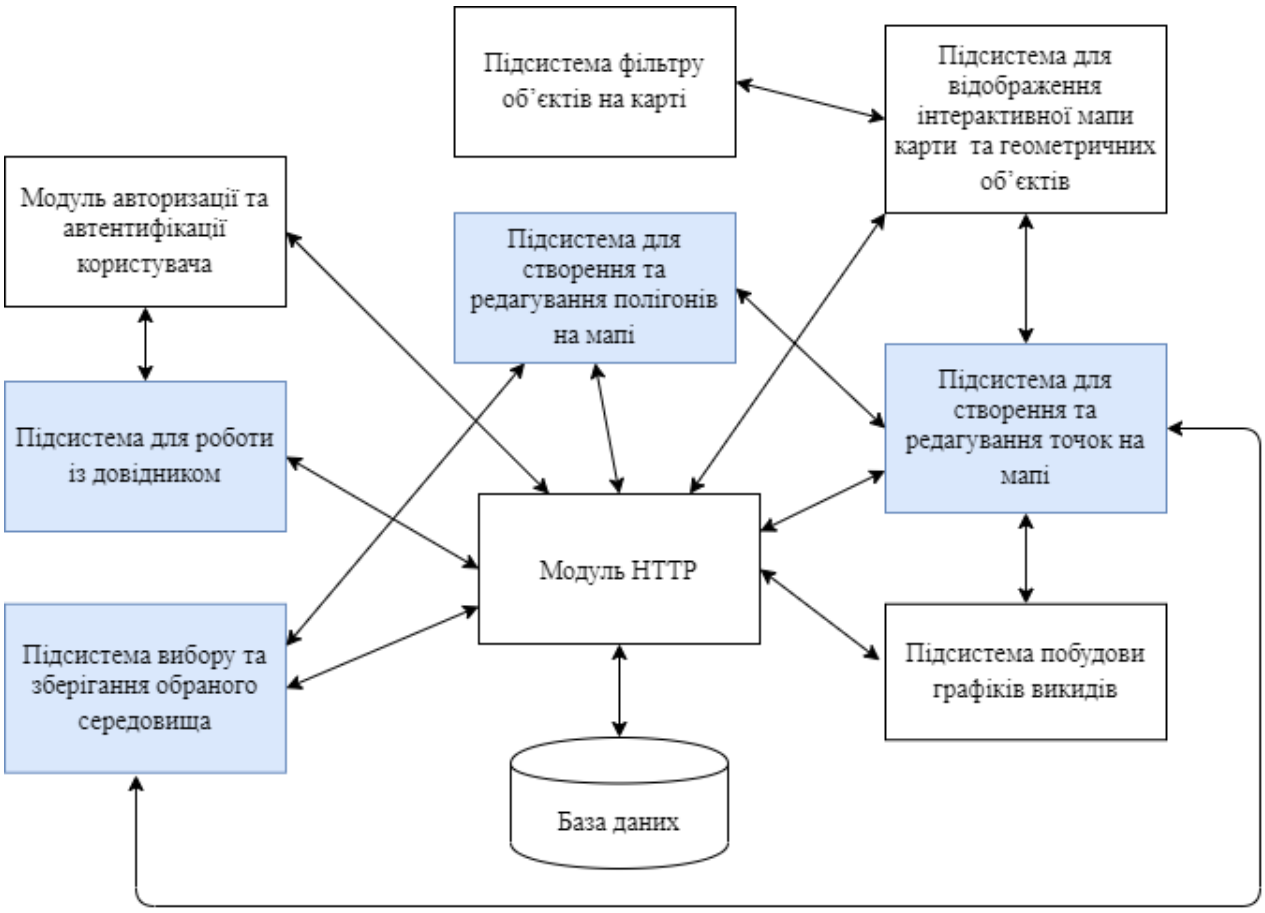


Рисунок 4.1.3 — Підсистеми для роботи із маркерами, полігонами та довідниками

Серверна частина відповідає за обробку запитів клієнта, комунікацію із реляційною базою даних, реалізацію функцій для роботи з даними: вставка даних, їх редагування та вибірка, а також відсилання оброблених даних до клієнта. Схема серверної частини зображена на рисунку 4.1.4.

У свою чергу її головними складовими є: конфігурація для роботи із базою даних, список API HTTP-шляхів для взаємодії із клієнтською частиною, контролери, як ланка взаємодії із БД.

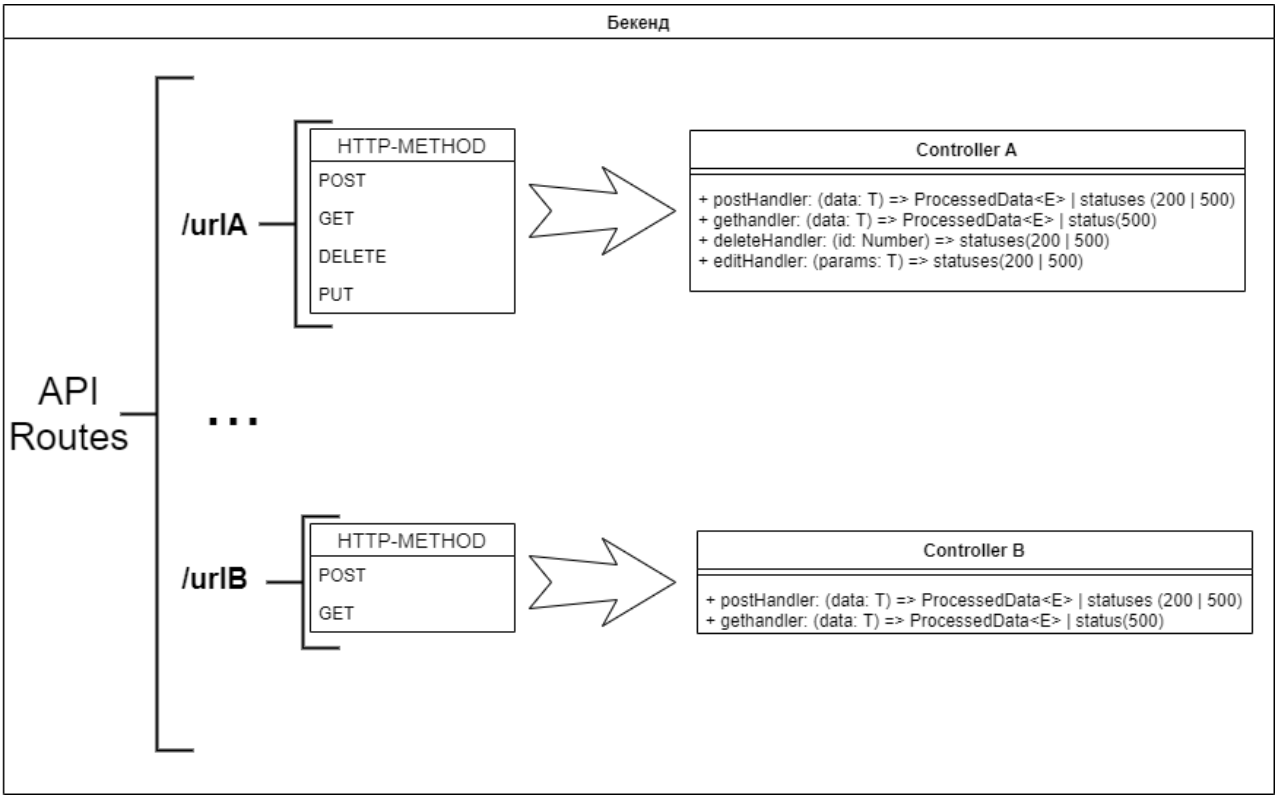


Рисунок 4.1.4 — Схема серверної частини

Конфігурація містить у собі інформацію, що необхідна для з’єднання із базою даних, а саме адресу бази даних, ім’я користувача, пароль та назву схеми. Додатково можна задавати порт та інші параметри з’єднання. У даній системі конфігурація знаходиться у файлі mysql-config.js і має наступний вигляд (рисунок 4.1.5):

Такий спосіб задання дозволяє швидко змінювати підключення. Іноді під час розробки виникала потреба змінити з’єднання із віддаленої бази даних на локальний сервер. Все, що для цього було необхідно зробити — вписати відповідні параметри у цьому файлі, після чого вся система почне працювати із локальною базою даних.

```
const mysql = require('mysql');
const pool = mysql.createPool({
  host: 'hostname',
  user: 'USER_NAME',
  password: 'PASSWORD_NAME',
  database: 'DATABASE_NAME',
});

module.exports = pool;
```

Рисунок 4.1.5 — Структура конфігурації для з’єднання із базою даних

У таблиці 4.1.1 наведено пояснення щодо полів даної конфігурації.

Таблиця 4.1.1. Параметри конфігурації для з’єднання

Назва поля	Пояснення
host	Вказується хост бази даних. Може бути як localhost, якщо сервер бази даних працює локально, так і IP адреса віддаленого серверу бази даних.
user	Ім’я користувача для входу
password	Пароль для входу
database	Необхідна база даних

До даного об’єкту конфігурацій можна включати й додаткові параметри: обмеження за кількістю з’єднань, можливість незахищеного доступу та інші.

Список API HTTP-шляхів для взаємодії із клієнтською частиною має наступну структуру:

router.<HTTP-method name>(<path>, <middleware?>, <controller handler function>);

Приклад такого зв’язку HTTP шляху та контролера зображено на рисунку 4.1.6.

```
router.post('/point', pointController.addPoint);
router.get('/point/:id', pointController.getPoint);
router.put('/point/:id', pointController.updatePoint);
```

Рисунок 4.1.6 — Приклад зв’язку HTTP шляху та контролера

Контролери взаємодіють із базою даних та підготовлюють дані для надсилання до клієнтської частини. Кожен із контролерів має відповідні методи, які пов’язують HTTP шлях API веб-системи із функціоналом, що взаємодіє безпосередньо із базою даних та обробляє результати. При запиті клієнта на конкретний шлях, що є у такому списку, на серверній частині спрацьовує відповідний оброблювач події контролера, який у результаті своєї роботи віддає підготовлені дані клієнтській частині та користувач бачить їх у себе у вікні браузера.

Контролер являє собою сутність, що містить у собі семантично закріплену абстракцію. Наприклад, контролер «point» відповідає за можливі дії із точкою, тобто додавання, читання, зміна та інші. З точки зору реалізації, це файл, всередині якого є функції, які приймають вхідні параметри, опрацьовують їх за допомогою бази даних і повертають результат.

У кожній із цих функцій контролера формуються відповідні MySQL-запити до бази даних. Ці запити можуть формуватися динамічно (рисунок 4.1.7).

```
SELECT
  id_of_user,
  poi.type,
  description,
  Name_object,
  owner_type as owner_type_id,
  owner_types.type as owner_type_name
FROM poi
INNER JOIN owner_types  ON poi.owner_type = owner_types.id
WHERE
  poi.Id = ${id}
;`;
```

Рисунок 4.1.7 — Приклад формування динамічного запиту

У випадку переліку параметрів, MySQL запити можуть формуватися параметрично (рисунок 4.1.8).

```
const updatedValues = { Name_object, description, type, owner_type };
const query = ` UPDATE ?? SET ? WHERE ?? = ? `;
const values = [tableName, updatedValues, 'Id', id];
```

Рисунок 4.1.8 — Приклад формування параметризованого запиту

Конструкції «??» та «?» використовуються для задання параметрів. Символи «??» застосовуються для виділення значень лапками, тобто як рядок, а «?» — певної обробки вхідних даних. Використовуючи символ «?» для різних типів вхідних даних буде застосовуватись різне форматування:

— для чисел (Number) формат зберігається;

— для логічних значень (Boolean) формат конвертується у true / false;

— для рядків (String) значення обертається у лапки;

— масиви (Array) конвертуються у список значень. Масив ['a', 'b'] перетвориться у 'a','b';

— вкладені масиви (Array<Array<T>>) групуються у списки. Приклад [['a', 'b'], ['c', 'd']] перетвориться у ('a', 'b'), ('c', 'd')

— об’єкти (Object) перетворюються у пари ключ — значення;

— null / undefined перетворюються у NULL;

У такий спосіб, можна легко конфігурувати SQL-запити, додаючи або видаляючи поля із JavaScript об’єктів та масивів. Це запобігає написанню великих запитів, які важко підтримувати у майбутньому та з високими ймовірностями помилок.

Вигляд параметризованого запиту, зображеного на рисунку вище, після конвертації бути мати наступний вигляд (рисунок 4.1.9).

```
UPDATE
        `<НАЗВА ТАБЛИЦІ>`
SET
        Name_Object = <Значення у об'єкті updatedValues з ключом 'Name_Object'>
        description = <Значення у об'єкті updatedValues з ключом 'description'>
        type = <Значення у об'єкті updatedValues з ключом 'types'>
        owner_type = <Значення у об'єкті updatedValues з ключом 'owner_type'>
WHERE
        `Id` = <Значення типу Number що знаходиться у змінній 'id'>
```

Рисунок 4.1.9 — SQL-запит після конвертації

Ці MySQL запити відправляються до віддаленого сервера бази даних. Після цього, результат обробляється і відправляється користувачу. Схема всієї бази даних має всього 31 таблицю із різними зв’язками між ними.

Логіка для введення показників та роботи із довідниками стосується не всіх таблиць. Таблиці, з якими в основному відбувається робота щодо внесення екологічних показників та взаємодії з довідниками, зображені на рисунку 4.1.9, проте дані, внесені до цих таблиць, напряму впливають на загальний стан системи.

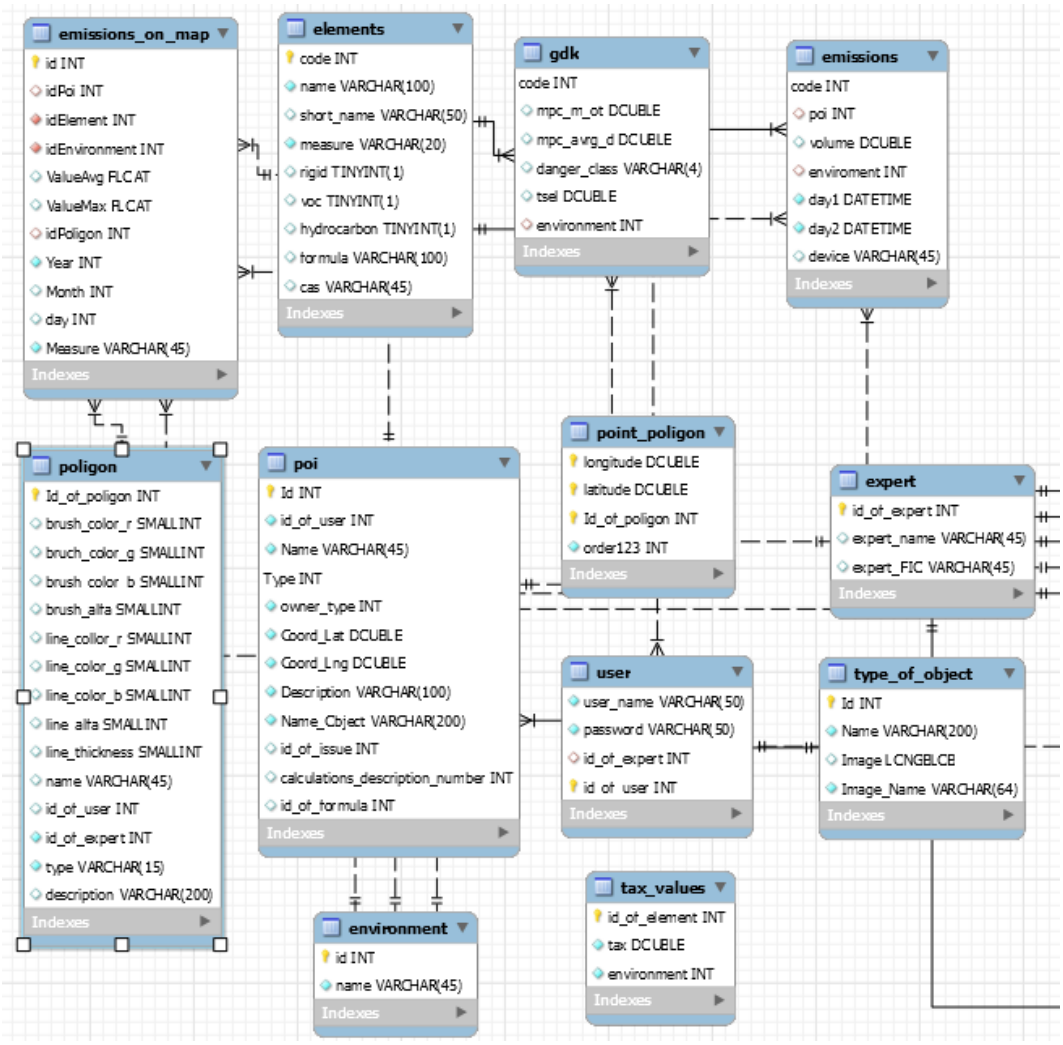


Рисунок 4.1.10 — Таблиці для введення показників та роботи із довідниками

Оскільки правильно спроектована база даних виключає надлишковість фактів у своїх таблицях, то всі дані мають бути розподілені по відповідних таблицях, що забезпечує швидкодію та відповідність потоку даних між таблицями.

На рисунку вище видно, що деякі поля відмічені ключем. Поняття «ключа» у базах даних широко відоме. Ключ — це мінімальний набір атрибутів, значення яких дозволяють однозначно обрати потрібний екземпляр сутності. Тут мінімальність вказується для того, щоб підкреслити, що усунення будь якого елементу із цього набору вже не дозволить ідентифікувати сутність. Ключі розділяють на різні категорії, проте однією із основних є обмеження цілісності даних, до якої відносять поняття первинного (англ. primary) та зовнішнього (англ. foreign) ключа. Первинний служить для однозначної ідентифікації сутності та обмеження цілісності у рамках

однієї таблиці. Первинний ключ не може бути пустим або повторюватись. Зовнішній ключ — це обмеження цілісності зв’язків декількох таблиць. Це дозволяє будувати цілісні моделі даних, оскільки дочірня таблиця не може посилатись на неіснуючі дані головної таблиці.

Як видно із рисунків, таблиці мають деякі зв’язки, що зображені лініями. Ці зв’язки створені для того, щоб пов’язувати між собою таблиці, дозволяючи збирати дані, що знаходяться у різних таблицях. Ці зв’язки бувають: один до одного (1:1), один до багатьох (1:n), багато до багатьох (n:n), який формується через пару зв’язків один до багатьох. Це дозволяє задавати відповідність полів одної таблиці до іншої. Приклад з’єднання один до багатьох зображено на рисунку 4.1.11, де одному експерту може відповідати декілька користувачів.

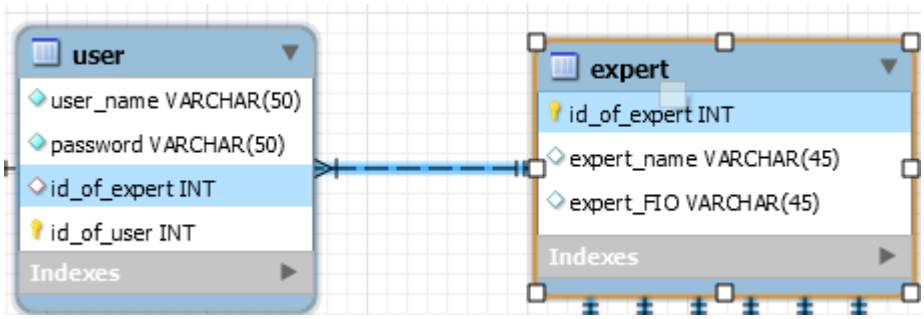


Рисунок 4.1.11 — Приклад зв’язку один до багатьох (1:n)

Комунікація з базою даних відбувається за допомогою спеціальної сутності «pool» NPM-пакета для роботи з MySQL. Для підвищення плавності взаємодії, використовуються методи асинхронного програмування, а саме конструкції Promise, що дозволяють обробляти дані асинхронно.

Задля швидкого виявлення можливих помилок, були додатково описані конструкції catch для обробки помилок. У разі виникнення помилки при роботі із базою даних чи при помилці сервера, на клієнтську частину відсилається відповідь про помилку на сервері з полем message, що містить детальну інформацію про помилку.

Клієнтська частина надає графічний інтерфейс користувача для забезпечення

комфортної роботи під час внесення та перегляду даних. Веб-система являє собою SPA (англ Single Page Application), що складається із головної сторінки, карти викидів та сторінки з довідниками.

Основну масу на клієнтській частині складають компоненти (рисунок 4.1.12).

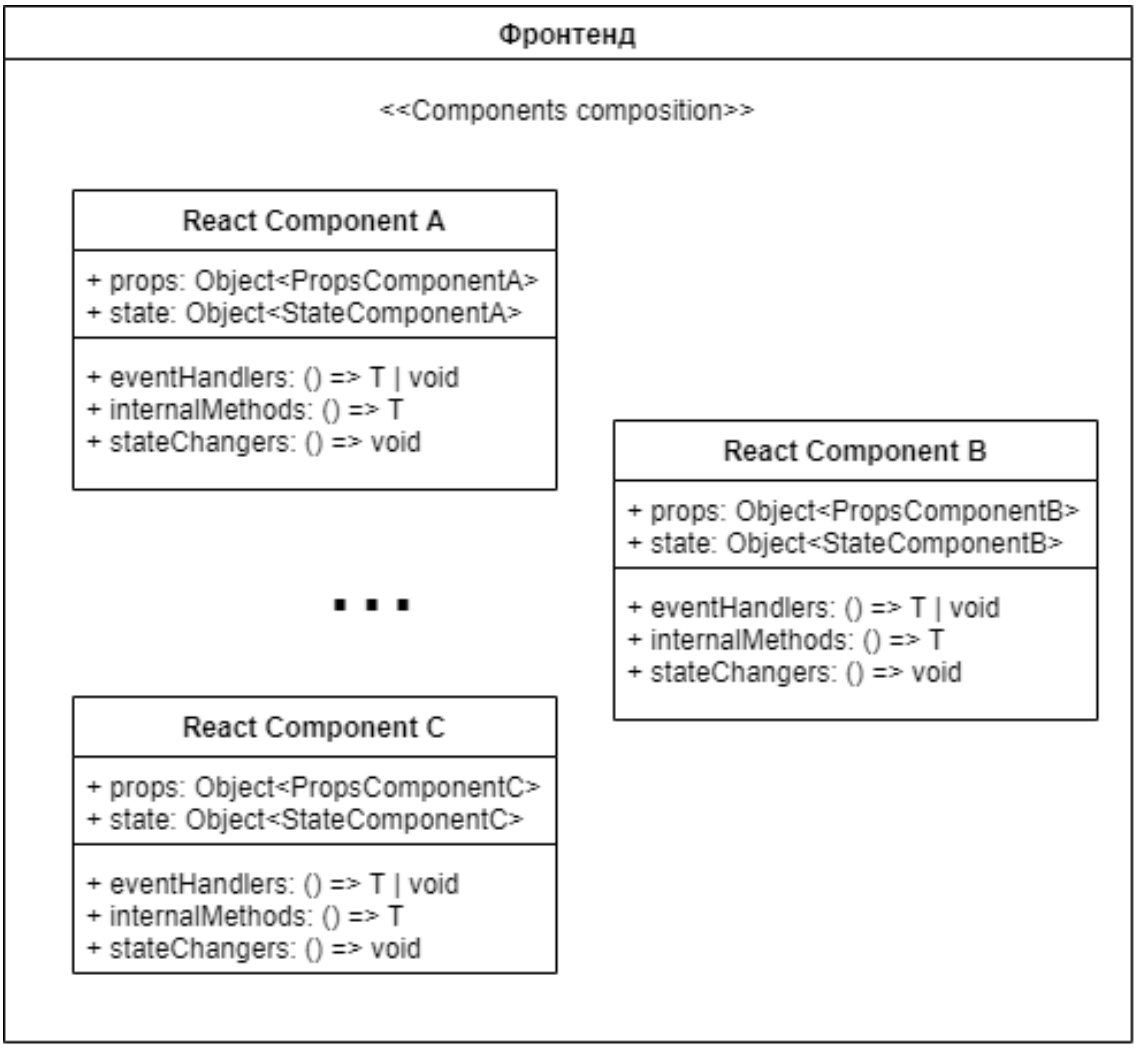


Рисунок 4.1.12 — Компоненти на клієнтській частині

Компонент — це атомарна, повністю закінчена частина коду з однозначно визначеною метою, що виконує поставлену задачу, інкапсулюючи у собі логіку функціоналу та взаємодіє з іншими компонентами через чітко визначений API. Компоненти взаємодіють між собою використовуючи «контракти», що вказують набір параметрів, які потрібно передати. щоб компонента функціонувала правильно. Оскільки було використано бібліотеку React для забезпечення швидкодії системи, то

додатково слід сказати про показ компонентів. Вхідні параметри, що надходять у компонент називаються «пропсами» (англ. props), вони можуть бути як звичайними даними, так і функціями. Ці параметри задають методи й значення, з якими компонент буде взаємодіяти та реагувати на зміни. Також це дозволяє задавати для одного компоненту різні дані. Компонент може зберігати у собі стан, що міняється у залежності від дій користувача або взаємодії даних. Зазвичай, зміна будь-якого «пропса» чи стану компоненти запускає процес її перемальовування. Отже, забезпечується реактивність даних.

Оскільки React при роботі використовує модель VirtualDOM, то при змінах він слідкує за тим, які компоненти змінились і які треба перемалювати, порівнюючи два стани дерева компоненти: початкового і зміненого. У такий спосіб, досягається швидкодія всієї системи, бо перемальовуються лише ті компоненти, які змінились. На рисунку 4.1.13 показаний приклад компоненти з «пропсами» і станом.

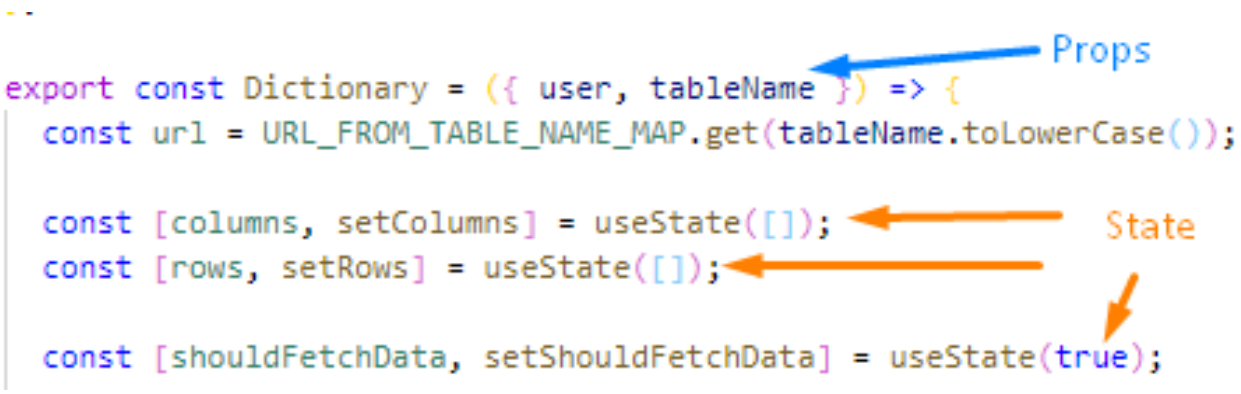


Рисунок 4.1.13 — Приклад компоненти із станом і «пропсами»

Компоненти можуть бути використані у будь-якому місці програми й самостійно функціонувати, якщо виконані усі умови «контракту».

Такий підхід дозволяє використовувати компоненти у різних місцях, не дублюючи код, та допомагає швидко виявляти помилки, оскільки логіка компоненти знаходиться тільки в одному місці. Фрагмент схеми компонентів клієнтської частини зображений на рисунку 4.1.14. Композиція таких компонентів формує складні інтерфейси веб-системи.

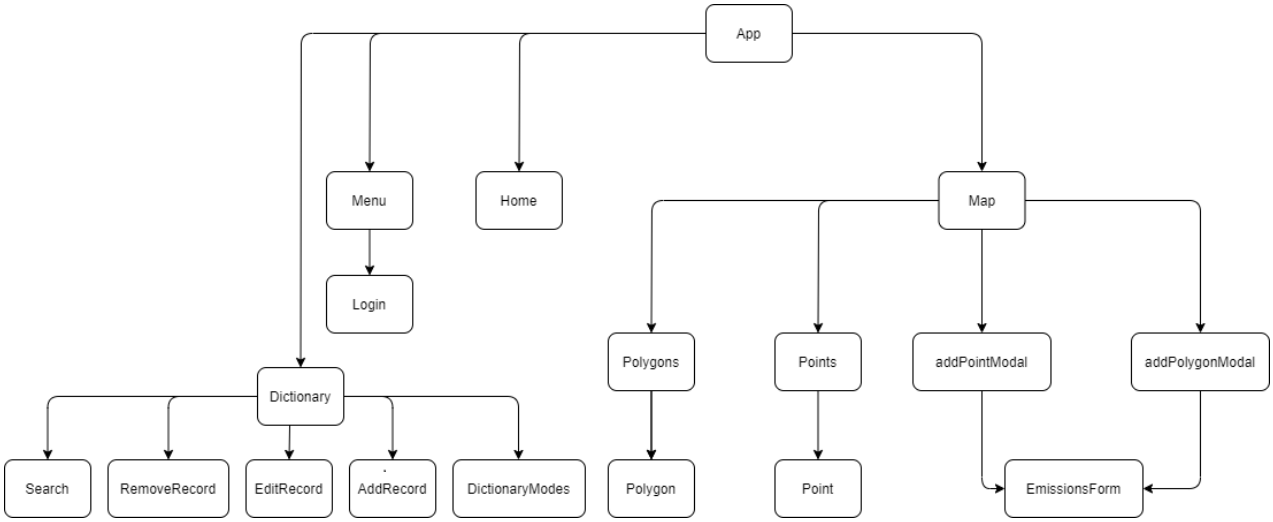


Рисунок 4.1.14 — Фрагмент схеми компонентів клієнтської частини

Додатково використовуються бібліотеки, що забезпечують взаємодію із картами та задання уніфікованого стилю усієї системи. Такими бібліотеками є react-leaflet і react-bootstrap відповідно. Для побудови довідників було використано бібліотеку Ag-Grid.

Окремим модулем є сервіс HTTP, що містить у собі логіку для здійснення HTTP запитів. Будь яка компонента, що робить запит на сервер обов’язково звертається до цього сервісу, який дозволяє зробити HTTP запит.

4.2. Взаємодія з маркерами та полігонами

Основними елементами на карті даної веб-системи є маркери та полігони. Маркерами на карті зображуються об’єкти, які можуть відповідати заводам, фабрикам, фермам та іншим структурам. Кожному маркеру задається іконка на карті, що дозволяє легко визначити приналежність даного об’єкту. Маркер має свій тип, ім’я, опис, форму власності, а також до нього може додаватись інформація про викиди речовин. Полігон — це область на карті, що охоплює певну територію і дозволяє зображати цілі ділянки місцевості. Полігону задаються: назва, опис, інформація про викиди, а також опції, що специфічні для полігону, як-от колір та товщина лінії.

Коли експерт додає чи редагує об’єкт, він ініціює HTTP запит до сервера. Сервер перехоплює цей запит і починає обробляти вхідні параметри. Обробивши їх та сформувавши відповідні MySQL запити для додавання чи редагування об’єкту, а також внесення викидів, він відправляє їх до бази даних. Після цього перевіряється результат, якщо операція пройшла успішно, то користувачу повертається позитивний статус та корисне навантаження. У разі виникнення будь якої помилки, сервер відправляє користувачу повідомлення про помилку.

Інформація про викиди складається із дати викиду, середнього та максимального значення викиду, елементу та обраного середовища. Це означає, що в об’єктів на карті викиди можуть бути із різних середовищ, таких як вода, ґрунт, атмосфера та інші. Для дати присутня валідація, що не дозволяє вказати майбутню дату. Якщо користувач намагається вказати майбутню дату, буде автоматично вибрано поточний день. При введенні викиду, у залежності від середовища та обраного елемента, відбувається синхронізація з наявними у базі гранично допустимими концентраціями. Фрагмент ГДК із бази даних зображений на рисунку нижче (рисунок 4.2.1).

code	name	short_name	measure	rigid	voc	hydrocarbon	formula	cas
0	Загальні викиди	Загальні викиди	мг/м³	1	0	0	-	-
1	Залізо та його сполуки	Залізо	мг/м³	0	0	0	Fe	-
2	Манган та його сполуки	Манган	мг/м³	0	0	0	Mn	-

Рисунок 4.2.1— Фрагмент ГДК із бази даних

Це дозволяє експерту проводити базовий аналіз отриманих даних ще на етапі введення. Якщо для вибраного елемента і середовища у базі даних не знайшлось гранично допустимих концентрацій, то ця перевірка буде проігнорована до моменту, поки вони не будуть внесені у систему.

Нижче подана діаграма діяльності процесу синхронізації введених даних та ГДК (рисунок 4.2.2). Діаграма ілюструє процес перевірки наявних гранично допустимих концентрацій та введених користувачем даних, з метою забезпечення можливості зробити первинний аналіз.

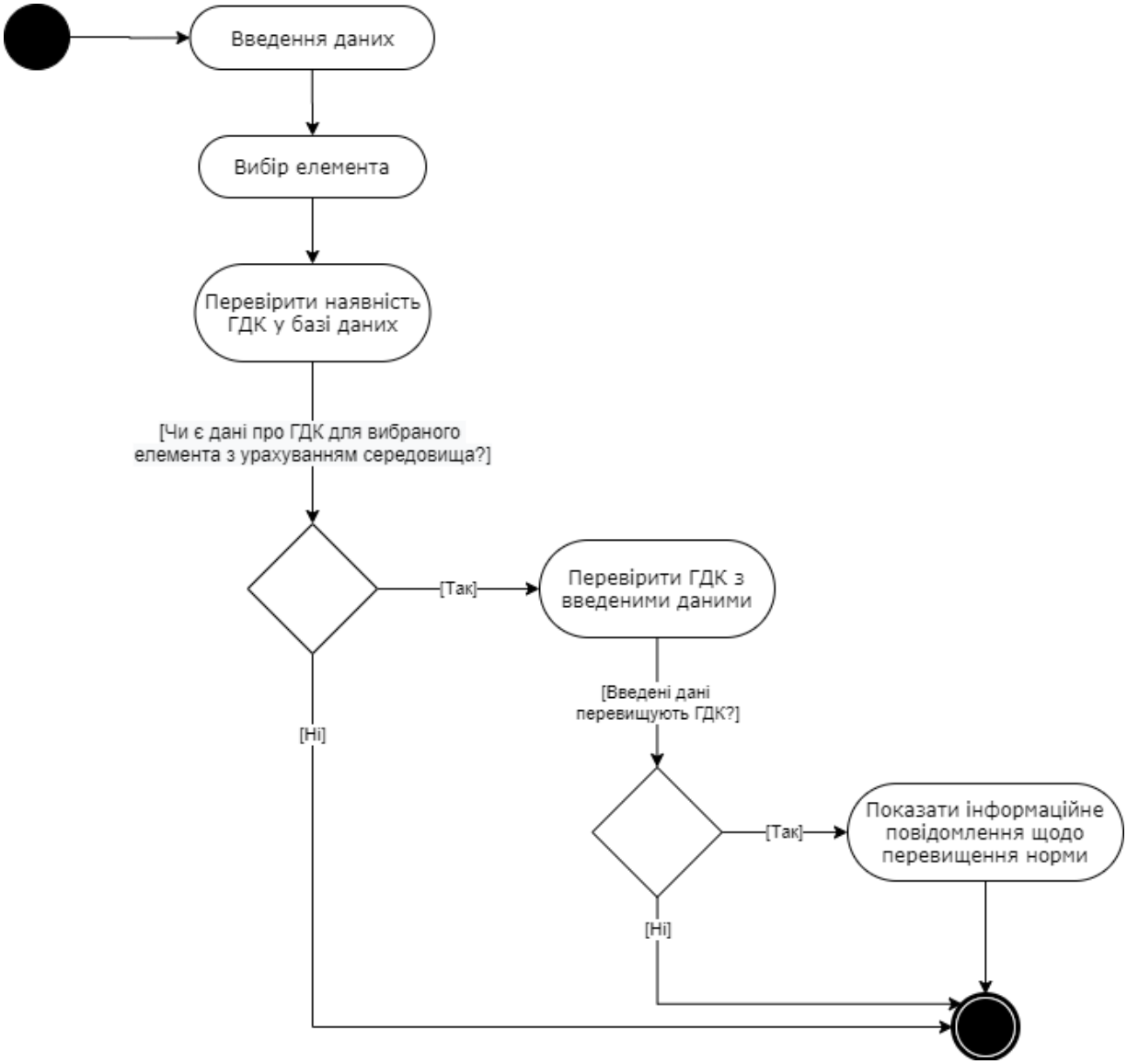


Рисунок 4.2.2 — Діаграма діяльності для процесу синхронізації ГДК і введених даних

Додатково, для заповнення даних було додано можливість імпорту з Excel та текстових файлів. Якщо у користувача є заповнений Excel чи текстовий файл із даними, то замість того, щоб самостійно вводити всі параметри, він може натиснути на кнопку завантаження даних із файлу і вони автоматично будуть підставлені у відповідні поля. Також, він може, тримаючи ліву кнопку миші, перетягнути файл із файлової системи у модальне вікно, де підставляться усі потрібні значення.

Оскільки формати даних для полігону і маркеру різні, то використовуються різні шаблони у файлах для імпорту, що дозволяє правильно заповнювати дані.

Структура файлу для заповнення даних про маркер вказана у таблиці 4.2.1.

Таблиця 4.2.1. Параметри файлу для заповнення маркера

Назва змінної	Пояснення
OBJECT_TYPE	Тип об’єкту. Додатково задає іконку маркеру
OWNER_TYPE	Форма власності
NAME	Назва маркеру
DESCRIPTION	Опис маркеру
DATE	Дата викиду
ELEMENT	Елемент, для якого задається інформація про викид
AVERAGE_VALUE	Середнє значення викиду. Динамічно порівнюється із ГДК.
MAXIMUM_VALUE	Максимальне значення викиду. Динамічно порівнюється із ГДК.

Структура файлу для параметрів полігону зазначена у таблиці 4.2.2.

Таблиця 4.2.2. Параметри файлу для заповнення полігону

Назва змінної	Пояснення
LINE_THICKNESS	Товщина лінії полігону
COLOR	Колір полігону. Вказується у форматі RGBA(0…255, 0…255, 0…255, 0…1)
NAME	Назва полігону
DESCRIPTION	Опис полігону
DATE	Дата викиду
ELEMENT	Елемент, для якого задається інформація про викид
AVERAGE_VALUE	Середнє значення викиду. Динамічно порівнюється із ГДК
MAXIMUM_VALUE	Максимальне значення викиду. Динамічно порівнюється із ГДК

Для перевірки коректності імпортованого файлу було додано обробку помилок. Якщо імпортований файл не відповідає формату, то показується помилка про неправильні дані. Наприклад, якщо користувач випадково завантажить файл із даними для полігону у модальне вікно, що стосується маркеру, програма сповістить про невірний формат даних та користувач зможе завантажити відповідний файл. Також для полігону проводиться перевірка правильності задання кольору і, якщо формат буде неправильним, то програма про це повідомить.

4.3. Функціонал роботи з довідниками

Оскільки було поставлено завдання для роботи із довідниками з розподіленими можливостями для адміністратора й інших користувачів, був розроблений функціонал, який це забезпечує. Для користувача доступні лише опції перегляду, пошуку даних по всіх можливих збігах, фільтрації та експорту у CSV. У свою чергу, адміністратор має ширші можливості для роботи із довідниками, тому він додатково може редагувати, додавати та видаляти дані. USE-CASE діаграма наведена на рисунку 4.3.1.

Ці операції напряму впливають на базу даних, тому такі опції доступні лише адміністратору. Крім того, при видаленні запису, додатково з’являється вікно для підтвердження операції. Якщо повідомлення для підтвердження було введено неправильно, то операція скасовується. Таким чином адміністратор усвідомлено видаляє дані та не зможе видалити їх випадково.

Окремо необхідно зазначити, що після операцій видалення, додавання та редагування з’являється вікно, яке показує результат виконання операції.

Якщо операція пройшла успішно, то вікно сповістить про це. У випадку помилки при здійсненні операції, адміністратор буде сповіщений про те, що відбулась помилка і буде подана інформація про саму помилку. Для виконання операцій додавання, редагування та видалення використовуються POST, PUT,

DELETE HTTP-запити відповідно.

Користувач може отримати інформацію про наступні довідники:

—елементи;

—ГДК (гранично допустимі концентрації);

—середовища;

—типи об’єктів;

—податки;

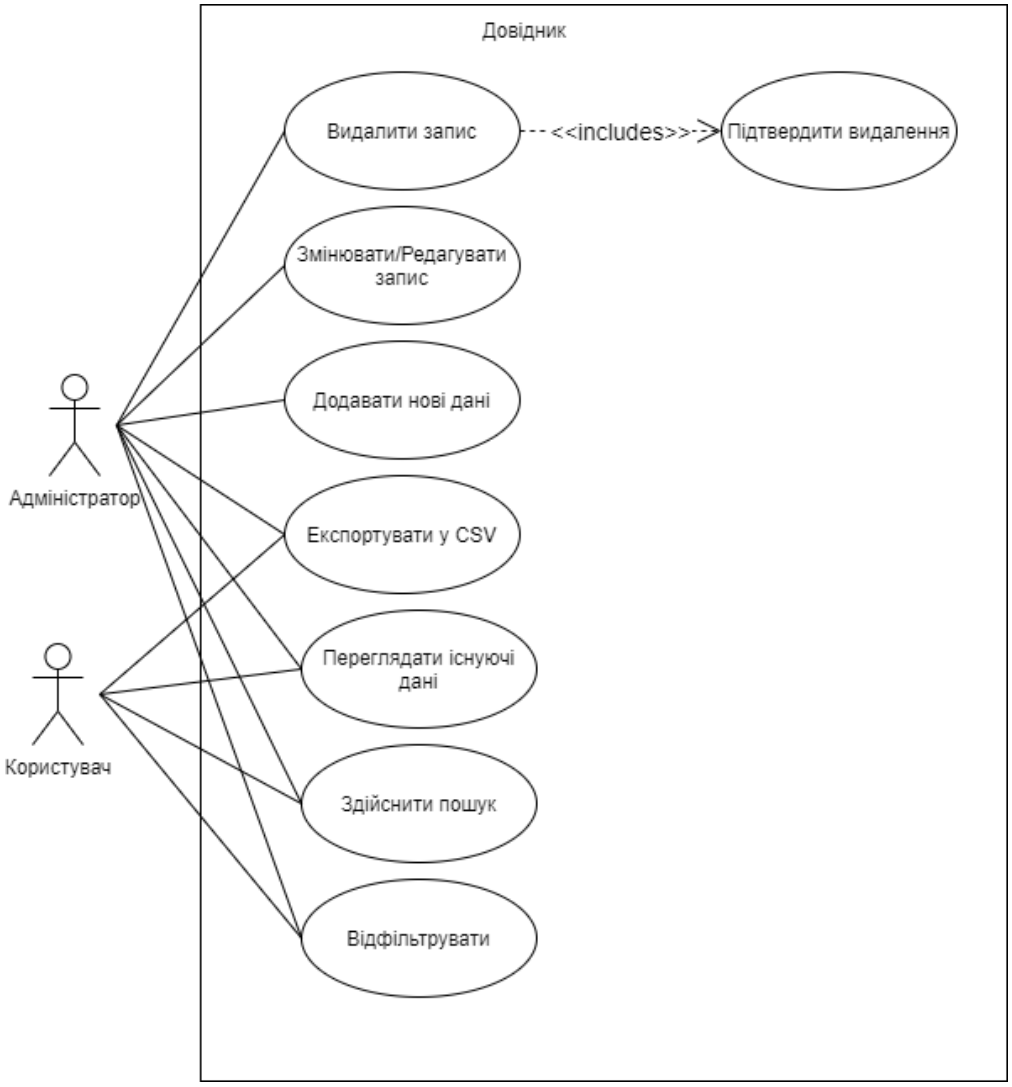


Рисунок 4.3.1 — Use-case діаграма для роботи із довідниками

При виборі будь-якого із цих довідників виконується GET-запит для отримання відповідної інформації. Після обробки даних, на сторінці користувача будується таблиця з отриманою інформацією із бази даних.

Для забезпечення більш гнучкої роботи із довідником, користувач має можливість сортувати дані по колонках, задавати фільтри окремим колонкам за деякими правилами, що дозволяє виключати/включати дані до вибірки. Якщо для кращого візуального представлення необхідно поміняти колонки місцями, то користувач може переміщати відповідну колонку, тримаючи ліву кнопку миші.

Над таблицею знаходиться основна панель для роботи із довідником. Вона містить опції для пошуку, додавання, редагування, видалення, а також кнопку експорту даних у форматі CSV. Загальна схема роботи із довідниками зображена на рисунку 4.3.2.

Вибираючи опції, користувач вмикає певний режим роботи із довідником. Опція пошуку розроблена таким чином, що пошук ведеться по всіх даних довідника та відразу при введенні. Отже, пошук відбувається по всіх можливих збігах і користувачу не потрібно натискати на додаткову кнопку пошуку. Додавання запису відбувається за допомогою внесення у поля потрібних даних. Редагування схоже до додавання, але поля заповнюються відповідно до обраного елементу для редагування. Для вибору елементу необхідно натиснути на потрібному рядку у довіднику. Вибраний рядок буде виділений синім кольором. При видаленні користувач повинен вибрати елемент, який він хоче видалити та підтвердити операцію. Після усіх цих дій з довідником, нові дані з’являються автоматично та без необхідності перезавантаження сторінки.

Окремою опцією для користувача та адміністратора є можливість експорту даних у CSV форматі. CSV — розшифровується як «comma-separated values», що являє собою текстовий файл, який складається зі значень, що, зазвичай, розділені комою. Цей формат є надзвичайно популярним для зберігання табличних даних та їхнього обміну. Даний формат можна перетворити у звичайний Excel і, наразі, нові версії Excel можуть правильно відкривати такі файли. Важливо зазначити, що опція експорту даних застосовується не для всього довідника, а лише для тих даних, що відфільтрував користувач. При цьому сортування елементів також зберігається.

Спосіб показу довідників і роботи з ними розроблений таким чином, щоб забезпечувався увесь функціонал і комфорт роботи із ними за допомогою мобільних

пристроїв.

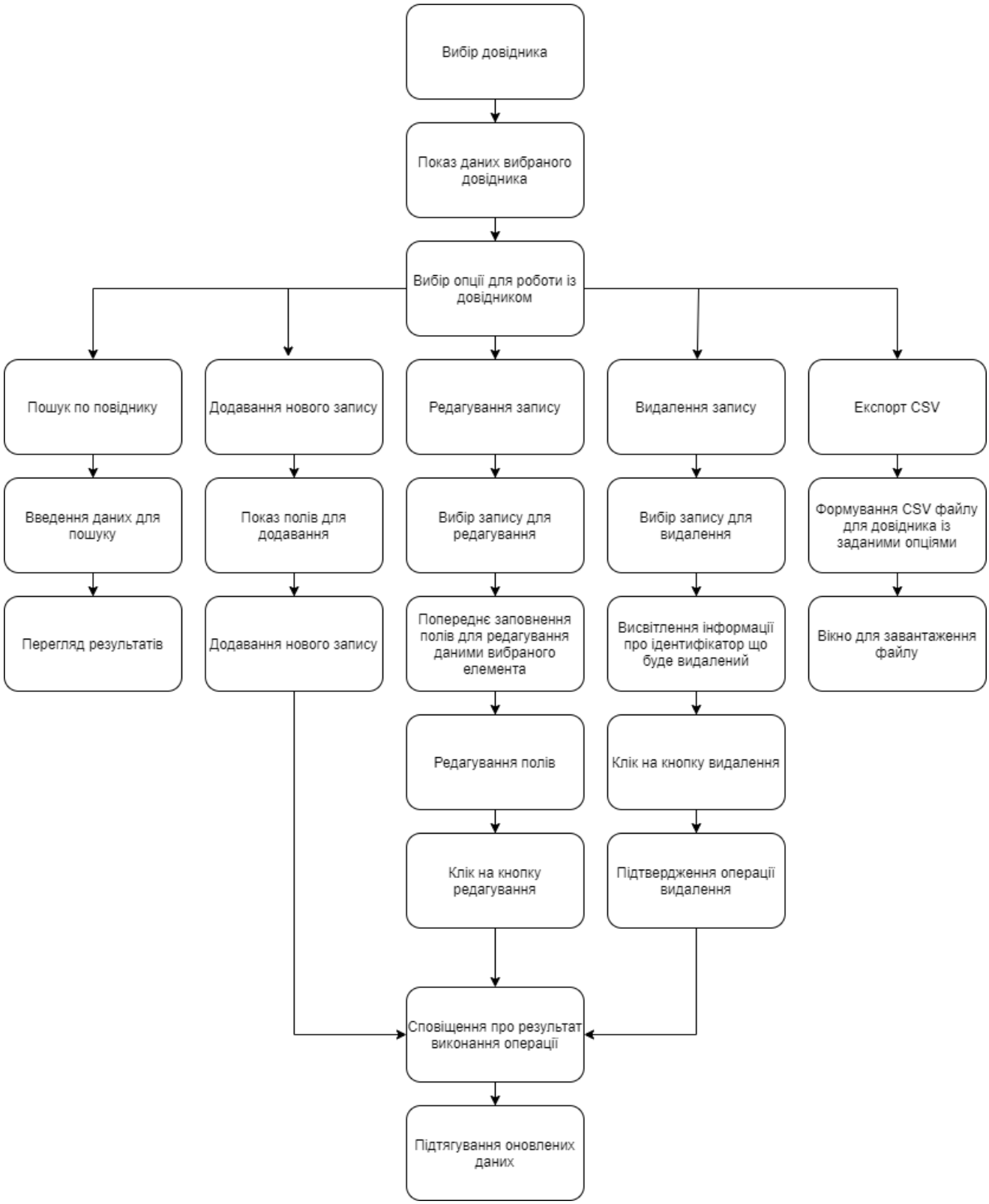


Рисунок 4.3.2 — Схема роботи із довідниками

У такий спосіб, були реалізовані функції для роботи з довідниками, що забезпечують виконання усіх необхідних операцій, поставлених у задачі, із

врахуванням прав користувачів.

4.4 Висновки до розділу

Результатом розробки веб-системи є програмний продукт, побудований на клієнт-серверній архітектурі, що забезпечує обмін даними між користувачем та базою даних. Було детально описано реалізацію функціоналу, що дозволяє вносити екологічні параметри до системи за допомогою карти. Для зручності внесення таких параметрів була додана можливість імпорту даних з Excel та текстових файлів.

Також був розроблений функціонал для роботи із довідниками. У довідниках присутній контроль доступу до функцій, який базується на правах користувачів. Опції довідника дозволяють шукати, додавати, видаляти, редагувати та експортувати дані. Після операцій із довідником, система відразу застосовує зміни й завжди працює в актуальному стані.

5. РОБОТА КОРИСТУВАЧА ІЗ ПРОГРАМНОЮ СИСТЕМОЮ

Оскільки система є веб-орієнтованою, вона не вимагає від користувача попередньої інсталяції. Для того, щоб почати користуватись системою, достатньо лише Інтернет з’єднання та браузера, який на сьогодні є на будь-якому смартфоні чи комп’ютері.

Після того як користувач переходить на сторінку веб-системи, він бачить головну сторінку з інформацією про комплексно еколого-економічний моніторинг та головним меню. Елементами меню є кнопка авторизації, карти зображень, довідники та повернення на головну сторінку (рисунок 5.1).

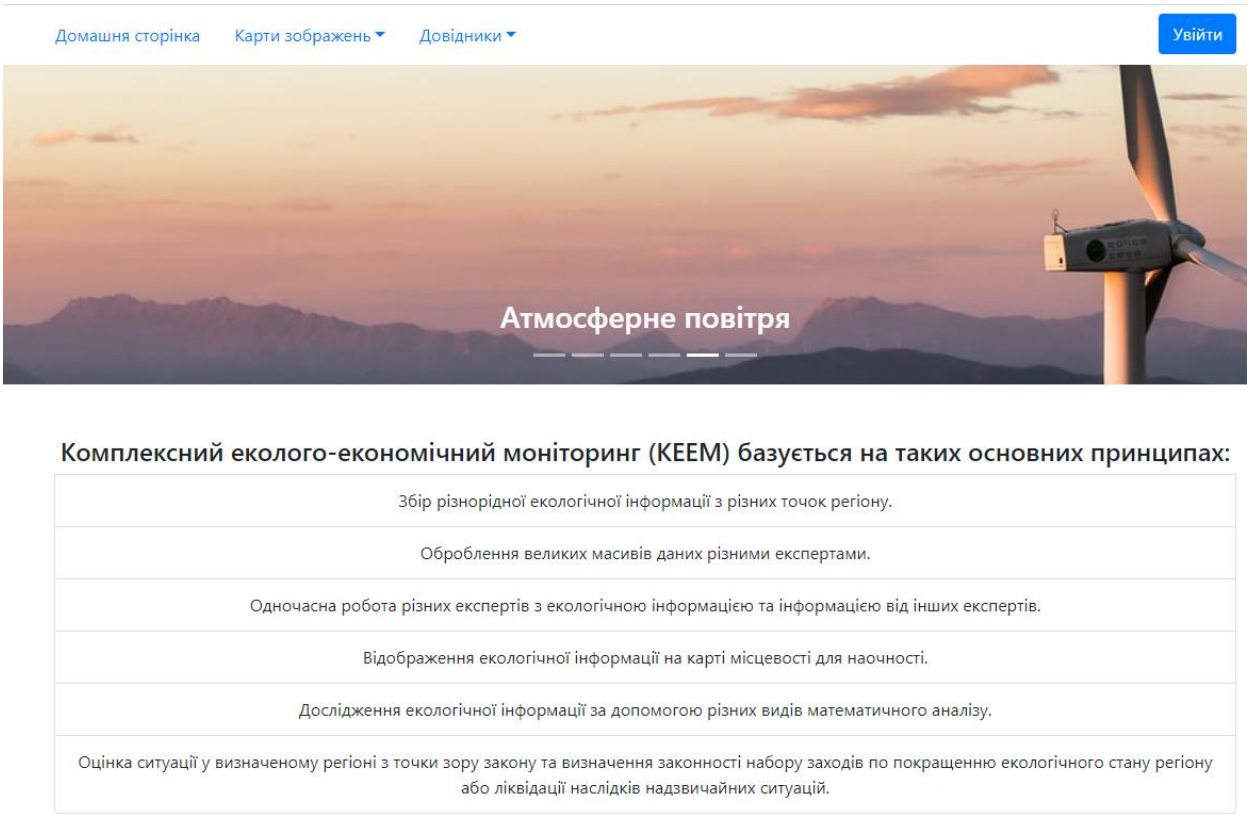


Рисунок 5.1 — Головна сторінка системи

Неавторизований користувач на карті може лише переглядати маркери й полігони та фільтрувати їх по експертах. Для того, щоб додавати позначення на карту, користувачу необхідно авторизуватись, використовуючи логін та пароль. Кожен із

користувачів являє собою експерта із визначеною роллю. Основними ролями є: еколог, медичний представник, економіст, юрист та особа, що приймає рішення.

Ключовою особою у веб-орієнтованій системі є еколог, який використовує карту для нанесення забруднювальних регіонів та об’єктів і вносить інформацію про обсяги викидів по різних показниках.

Для авторизації необхідно натиснути кнопку входу, після чого з’явиться модальне вікно для введення логіна та пароля. Форма авторизації зображена на нижче (рисунок 5.2).

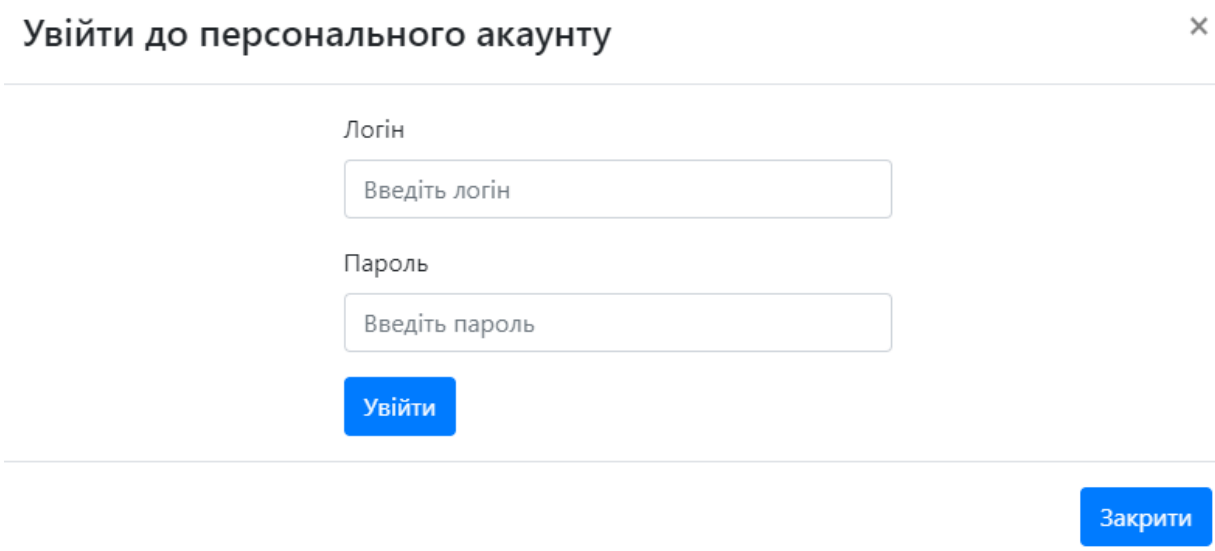


Рисунок 5.2 — Авторизація користувача

Після авторизації користувач побачить своє ім’я та роль у правому верхньому куті (рисунок 5.3).

Рисунок 5.3 — Авторизований користувач

За необхідності, користувач може вийти зі свого акаунту за допомогою спеціальної кнопки виходу.

Основна робота із даною системою являє собою взаємодію із картою, на якій

наносяться позначення різного типу. Загальний вигляд карти зображено на рисунку 5.4. Вибір карти зображень задає середовище, для якого будуть вноситись викиди.

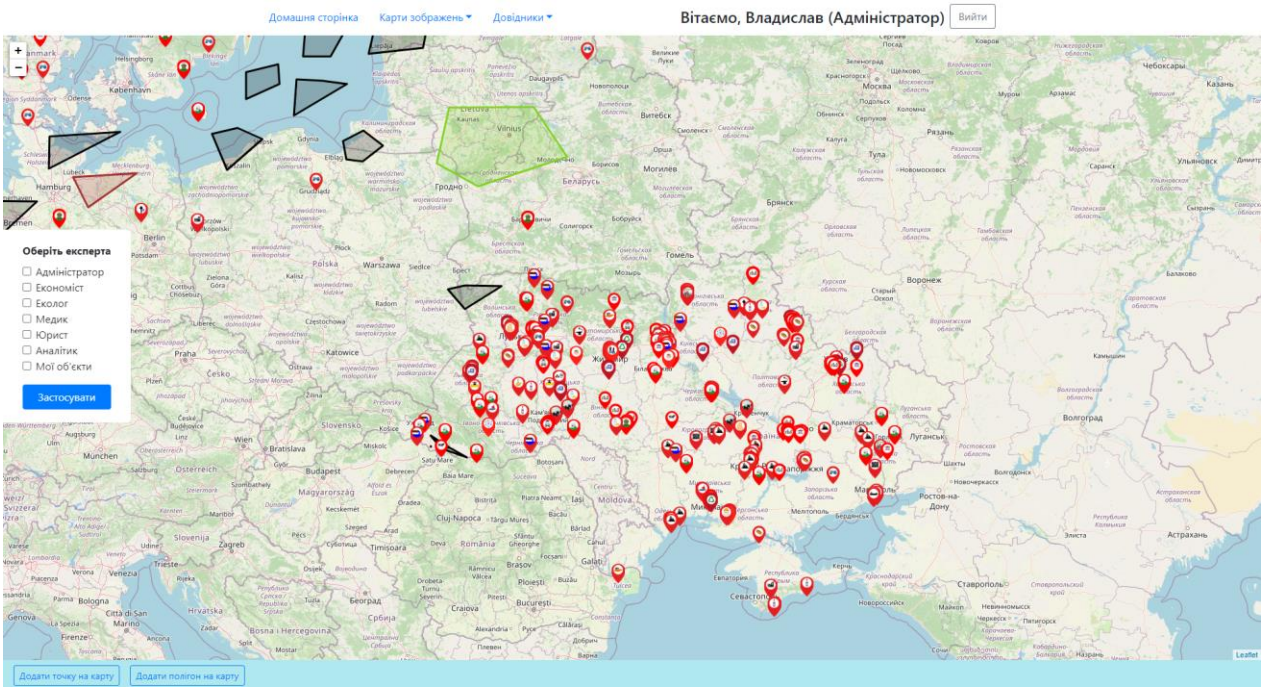


Рисунок 5.4 — Загальний вигляд карти

Для позначення точкового об’єкту, на карту можна додати маркер. Для виділення цілої області — полігон. Ці позначення задаються особами із визначеними ролями.

Для того, щоб ввести дані про новий екологічний показник, у лівому нижньому куті екрана слід обрати опції додавання маркера на карту або полігону. Після чого, на карті включається режим додавання (рисунок 5.5).

Рисунок 5.5 — Включений режим додавання

У випадку додавання маркеру, слід натиснути на відповідну кнопку додавання та вказати на карті місце, для якого будуть вводитись екологічні показники. Після цього з’явиться відповідне модальне вікно для задання параметрів.

Фахівець може обрати тип об’єкту та форму власності, вказати назву маркеру та його опис (рисунок 5.6).

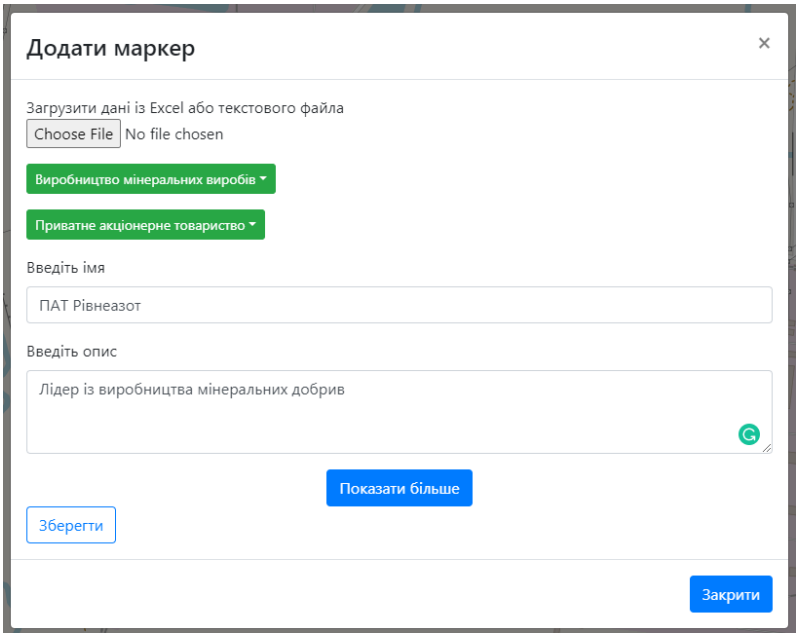


Рисунок 5.6 — Модальне вікно для додавання маркеру

За необхідності, можна відразу додати викиди. Для цього потрібно натиснути кнопку, яка дозволяє додати більше деталей до цієї точки (рисунок 5.7).

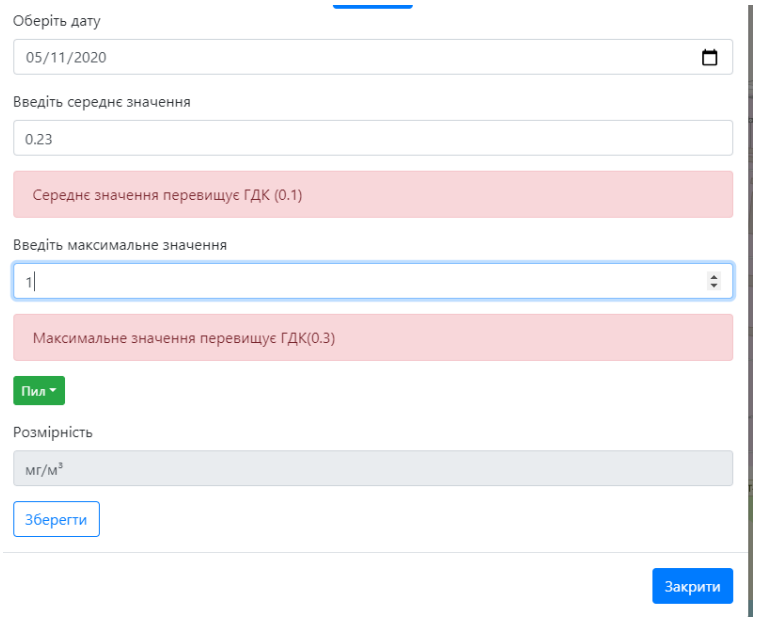


Рисунок 5.7 — Додавання викиду до маркеру

При додаванні викиду, середовище автоматично береться у залежності від вибраної користувачем карти зображень. У цьому додатковому меню, можна обрати дату, елемент, середнє і максимально значення викиду. При виборі елементу, введені значення будуть порівнюватись з наявними у базі даних. У випадку перевищення середнього та/або максимального значень викиду, користувач відразу побачить відповідні повідомлення.

У випадку додавання області на карту, користувач може обрати опцію додавання полігону. Це дозволить виділити область на карті, послідовно додаючи точки полігону. Після виділення необхідної області, з’явиться модальне вікно для внесення параметрів області (рисунок 5.8). У цьому модальному вікні задаються як параметри самого полігона, такі як: колір області, товщина лінії, назва та опис, так і додатково параметри викиду, де слід обрати дату, елемент і максимальне та середнє значення викиду, що будуть відразу порівняні з наявними ГДК.

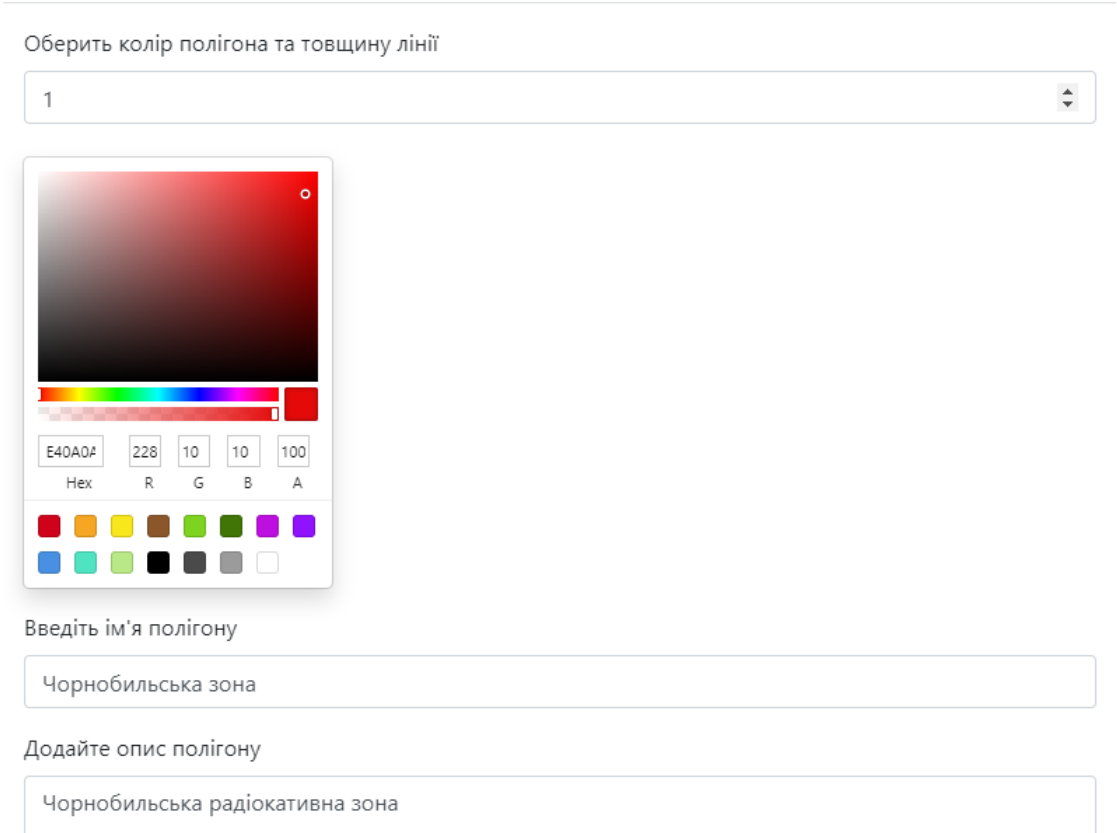


Рисунок 5.8 — Додавання полігону

Якщо фахівець бажає змінити дані про маркер, то для цього він натискає по

відповідному маркеру, де з’являється підказка з короткою інформацією про цей об’єкт. Для редагування використовується символ олівця, після натискання якого, відкриється модальне вікно з даними про цю точку. Аналогічно до маркерів фахівець може редагувати полігон, також натиснувши на іконці олівця у підказці для полігону. Додатково можна ввести інформацію про новий викид, як у випадку із маркером.

Для додавання або редагування полігону та маркеру користувач може скористатись опцією імпорту даних з Excel або текстового файлу. Для цього лише необхідно мати відповідний формат файлу і натиснути кнопку імпорту даних на модальних вікнах пов’язаних із маркером чи полігоном. Приклад Excel файлу для імпорту параметрів маркеру зображено на рисунках 5.9 та 5.10.

	A	B
1	OBJECT_TYPE	Забір, очищення та постачання води
2	OWNER_TYPE	Комунальна
3	NAME	Київводоканал
4	DESCRIPTION	Київський водоканал
5	DATE	28.12.2015
6	ELEMENT	Барий оксид
7	AVERAGE_VALUE	4
8	MAXIMUM_VALUE	3

Рисунок 5.9 — Приклад даних Excel файлу для заповнення маркеру.

Важливо додати, що при додаванні екологічних показників експертом, інші користувачі також побачать оновлений стан карти.

Окремою вкладкою є довідники (рисунок 5.11). Після натискання на цю вкладку, з’являється меню для вибору довідника, де буде продовжуватись робота. При переході до довідників є перевірка прав користувача, де з’ясовується чи є поточний користувач адміністратором, оскільки неавторизовані користувачі та користувачі, що не є адміністраторами мають обмежений список опцій для роботи із довідниками.

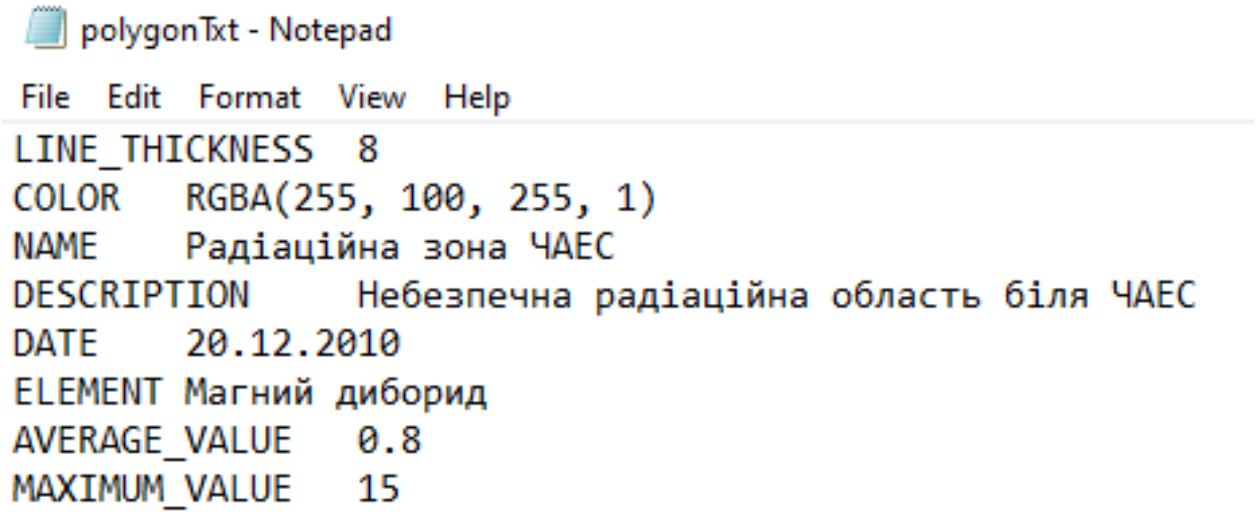


Рисунок 5.10 — Приклад даних текстового файлу для імпорту.

Вони можуть бачити дані довідника, фільтрувати та експортувати у CSV файл. Інші функції для роботи із довідником для них заборонені.

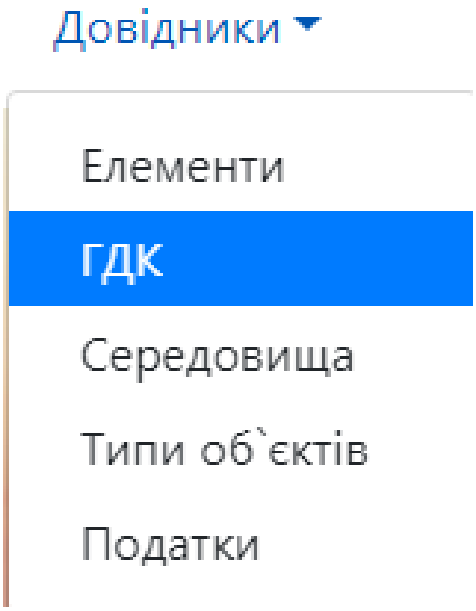
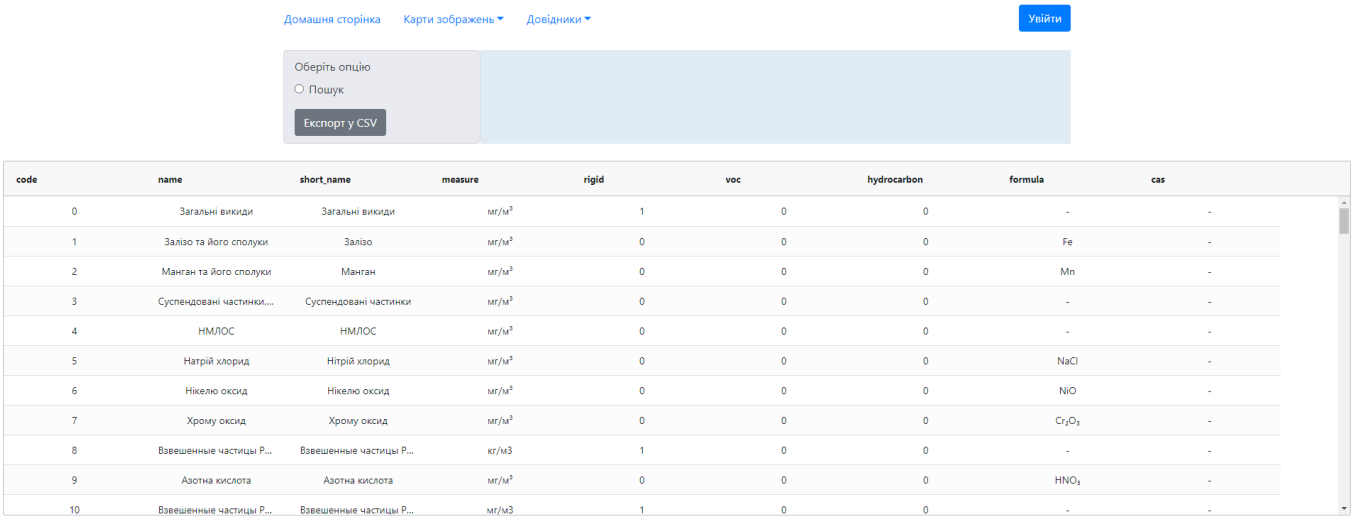


Рисунок 5.11 — Меню вибору довідників.

На рисунку 5.12 зображений вигляд системи для неавторизованого користувача з доступними йому опціями. Аналогічний вигляд будуть мати авторизовані користувачі, що не є адміністраторами.



code	name	short_name	measure	rigid	voc	hydrocarbon	formula	cas
0	Загальні викиди	Загальні викиди	мг/м³	1	0	0	-	-
1	Залізо та його сполуки	Залізо	мг/м³	0	0	0	Fe	-
2	Манган та його сполуки	Манган	мг/м³	0	0	0	Mn	-
3	Суспендовані частинки,...	Суспендовані частинки	мг/м³	0	0	0	-	-
4	НМЛОС	НМЛОС	мг/м³	0	0	0	-	-
5	Натрій хлорид	Нітрій хлорид	мг/м³	0	0	0	NaCl	-
6	Нікелю оксид	Нікелю оксид	мг/м³	0	0	0	NiO	-
7	Хрому оксид	Хрому оксид	мг/м³	0	0	0	Cr_2O_3	-
8	Взвешенные частицы Р...	Взвешенные частицы Р...	кг/м3	1	0	0	-	-
9	Азотна кислота	Азотна кислота	мг/м³	0	0	0	HNO_3	-
10	Взвешенные частицы Р...	Взвешенные частицы Р...	мг/м3	1	0	0	-	-

Рисунок 5.12 — Довідник для неавторизованого користувача.

Користувач може сортувати колонки, задавати для них фільтрацію, переміщати їх місцями для більш комфортного представлення даних, а також здійснювати пошук. Для того, щоб відсортувати дані довідника по деякій колонці, достатньо натиснути на назву цієї колонки у таблиці, після чого з’явиться стрілка, яка показує чи сортування відбувається за зростанням чи спаданням (рисунок 5.13).

id ↓	≡ name
6	Поверхневі води
5	Господарський грунт
4	Грунт
3	Технічна вода
2	Питна вода
1	Атмосфера

Рисунок 5.13 — Сортування даних по колонці.

Також можна задавати фільтри для окремих колонок, для цього необхідно натиснути на іконці «бургера» справа від колонки, після чого відкриється меню, у

якому можна задати фільтри для цієї колонки. Приклад таких фільтрів зображено на рисунку 5.14. Після задання таких фільтрів, відразу з’являється іконка фільтрів справа від імені колонки.

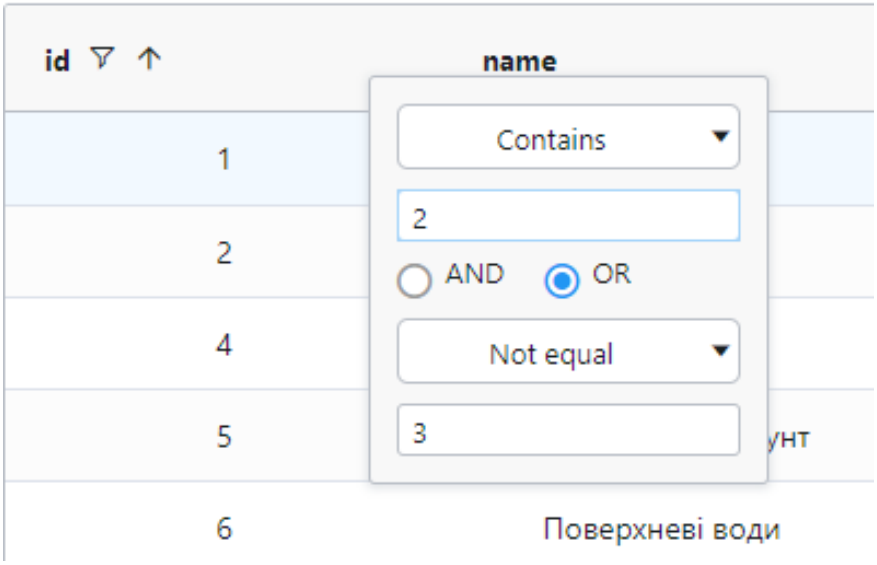


Рисунок 5.14 — Задання фільтрів для окремої колонки.

Якщо потрібно поміняти колонки місцями, достатньо натиснути ліву кнопку миші і, не відпускаючи її, перетягнути колонку на потрібне місце (рисунок 5.15).

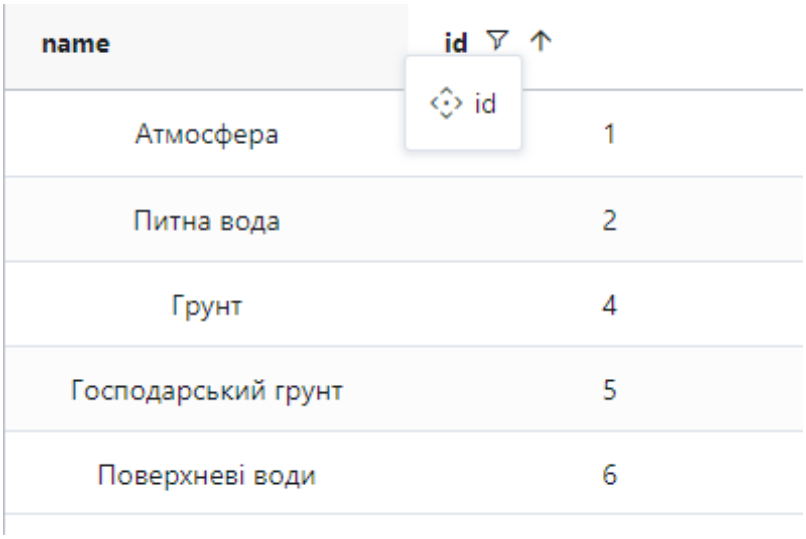


Рисунок 5.15 — Переміщення колонки.

Опція пошуку доступна усім можливим користувачам системи. Пошук

відбувається по усіх полях у довіднику, тобто система шукає усі можливі збіги. Оскільки пошук «живий», то для цього не потрібно натискати ніяких додаткових кнопок і, як тільки користувач вводить дані для пошуку, то інформація довідника відразу починає порівнюватись із пошуковим запитом (рисунок 5.16).

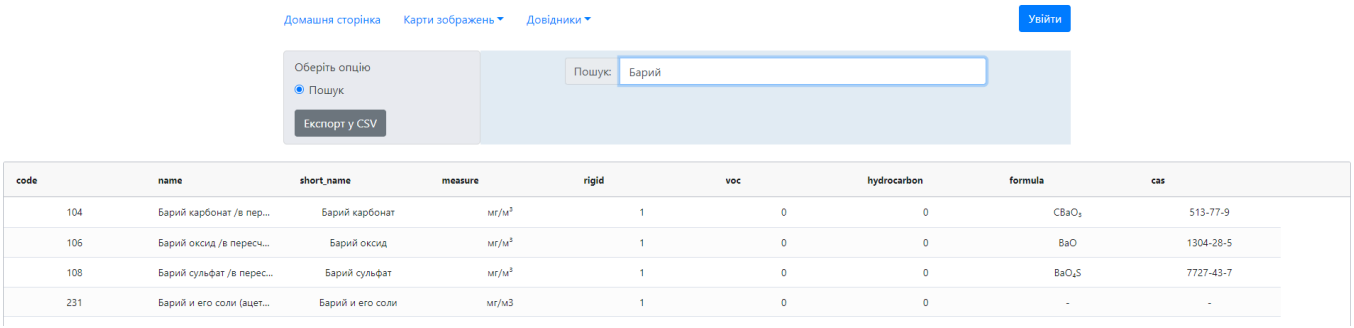


Рисунок 5.16 — Пошук по довіднику

У свою чергу, адміністратори мають більший набір опцій для взаємодії із довідниками. Оскільки адміністратори мають більше повноважень, то для них додаються опції додавання, редагування та видалення записів із довідника. Вигляд інтерфейсу для адміністратора має наступний вид (рисунок 5.17).

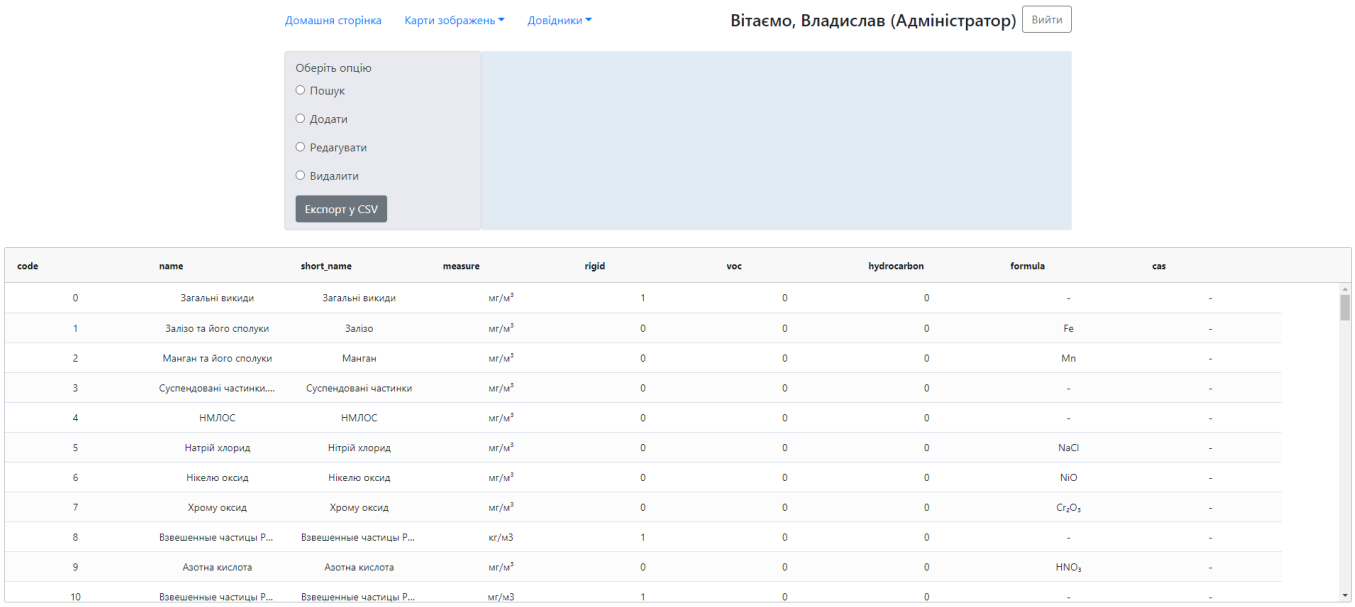


Рисунок 5.17 — Інтерфейс довідника для адміністратора

Вище представлена панель для роботи із довідником з усіма можливими опціями.

При виборі опції додавання в панелі управління, з’являється форма з полями для заповнення нового запису (рисунок 5.18).

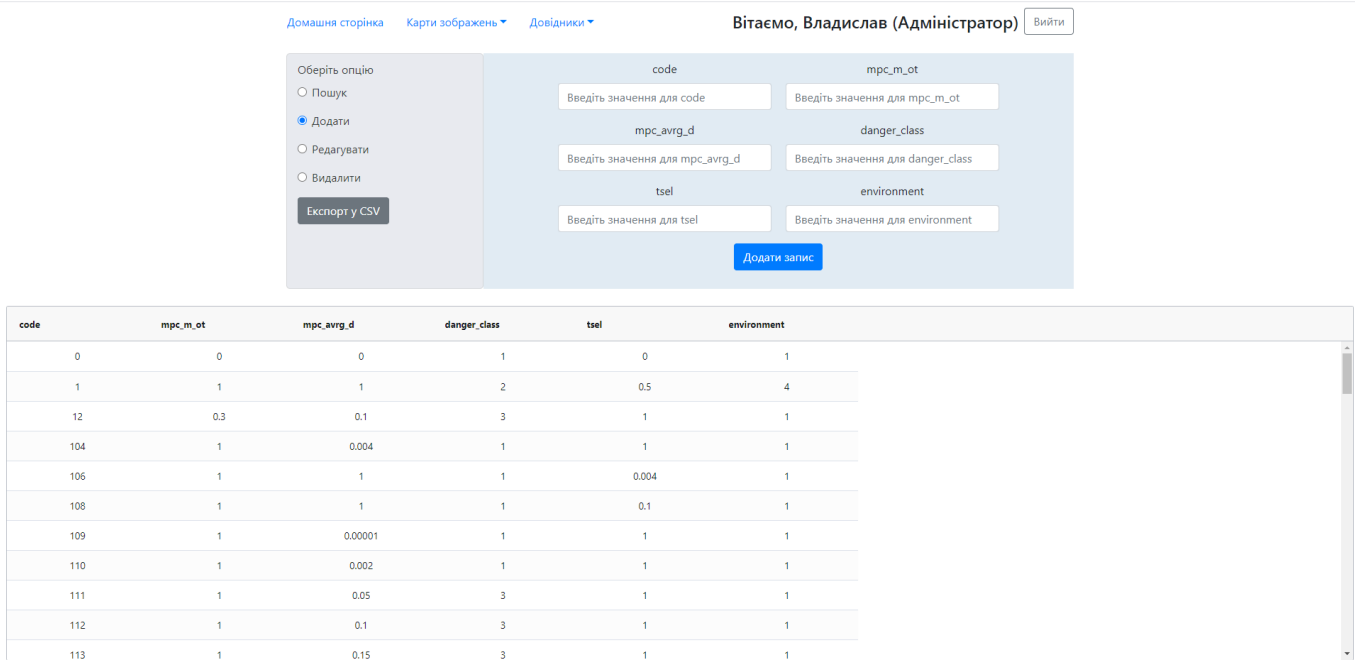


Рисунок 5.18 — Форма додавання нового запису

Після додавання запису, показується вікно зі сповіщенням результату виконання операції. У випадку успішного додавання запису, адміністратор буде повідомлений про це. При помилці виконання операції адміністратор побачить повідомлення про невдалу операцію і додатково отримає інформацію про помилку. Приклад такої помилки зображено на рисунку 5.19.

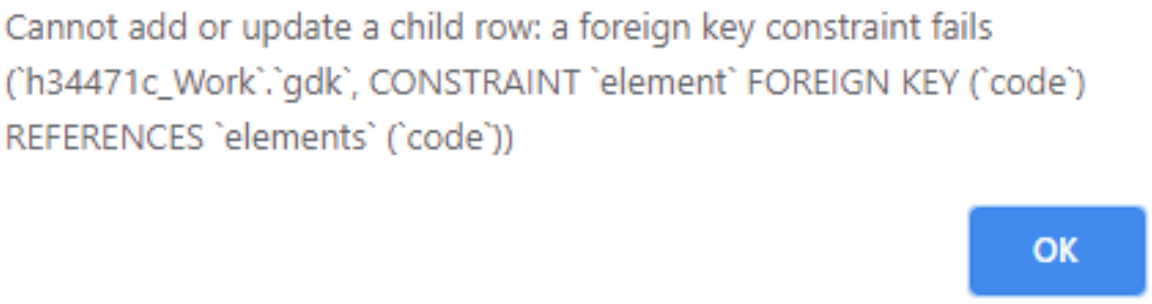


Рисунок 5.19 — Помилка при невдалій операції

Схожий алгоритм відбувається при редагуванні за виключенням того, що для початку потрібно обрати запис і потім відповідні поля заповнюються інформацією із вибраного запису, які, за необхідності, можна змінити.

При видаленні необхідно обрати елемент для видалення, після чого на панелі користувач побачить ідентифікатор запису, який буде видалено і кнопку видалення. Після натискання цієї кнопки відкриється вікно для підтвердження операції, де потрібно ввести ідентифікатор запису (рисунок 5.20).

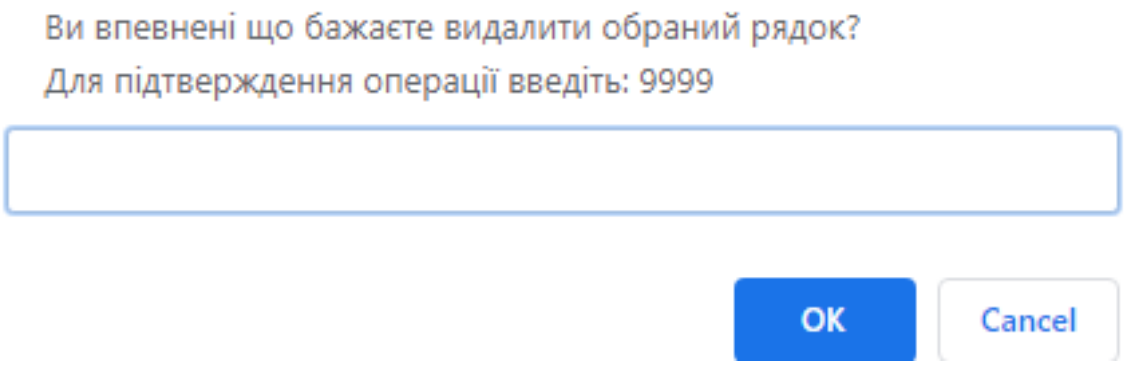


Рисунок 5.20 — Вікно підтвердження операції видалення

При правильному підтвердженні зініціюється операція видалення запису, після виконання якої буде показано результат.

При будь-яких змінах довідника, користувач буде бачити відразу оновлені дані, оскільки це значно покращує комфорт користування системою.

Як було зазначено раніше, окремою кнопкою є можливість експорту даних у файл CSV. Такий функціонал був доданий через потенційну необхідність експорту даних в інші програми або печать. При цьому експорт даних застосовується не всього довідника, що може бути не комфортним для користувача, а лише тієї інформації, що він має на екрані. Іншими словами, завантажуються дані уже із застосованими фільтрами, сортуваннями та результатами глобального пошуку. Це дає гнучкі можливості для користувача при роботі із довідником. На рисунку 5.21 наведено приклад такого експорту у CSV файл для довідника «Середовища» зі спадним сортуванням колонки «id» та глобальним пошуком по слову «Грунт».

	A	B
1	id	name
2	5	Господарський грунт
3	4	Грунт
4		

Рисунок 5.21 — Експорт у CSV

Оскільки для роботи із системою необхідно мати лише мережу Інтернет, було розглянуло питання щодо потенційного використання даної системи за допомогою мобільного телефону або планшету. Враховуючи це, була додатково виконана робота для забезпечення комфортної взаємодії користувачів з мобільних пристроїв Інтерфейс, при використанні мобільного пристрою, забезпечує весь функціонал, що є у повній веб-версії, а також автоматично підлаштовується під розмір екрану смартфона чи планшета користувача. Зовнішній вигляд карти на мобільному пристрої зображений на рисунку 5.22.

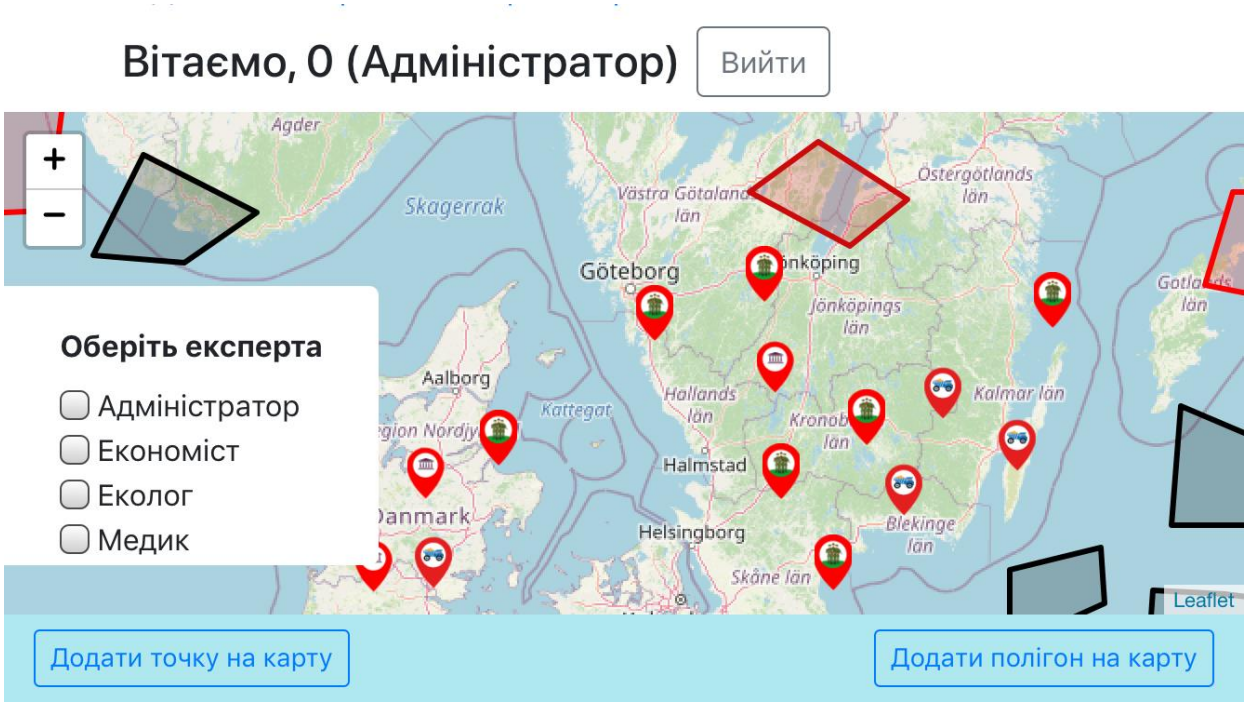


Рисунок 5.22 — Зовнішній вигляд карти на мобільному пристрої.

Користувач аналогічно може додавати або редагувати точки та полігони (рисунок 5.23).

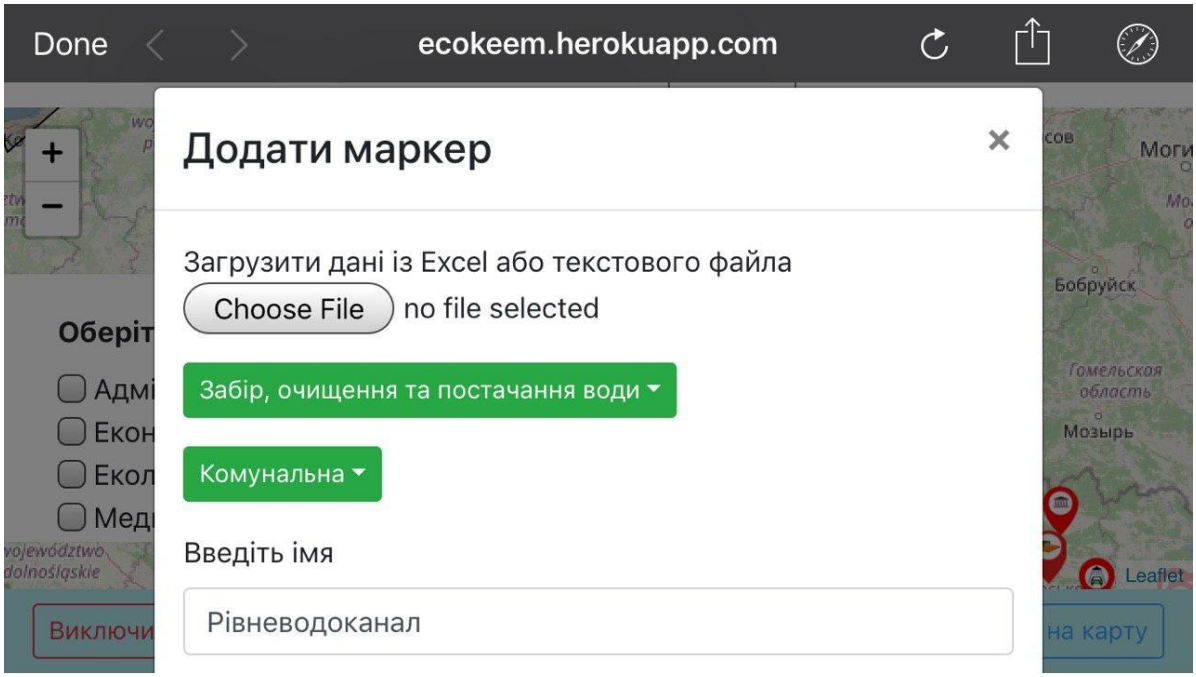


Рисунок 5.23 — Приклад додавання даних із мобільного пристрою.

Також, весь функціонал для роботи із довідниками доступний на мобільних пристроях, що дозволяє використовувати смартфон для швидкої взаємодії з ними (рисунок 5.24).

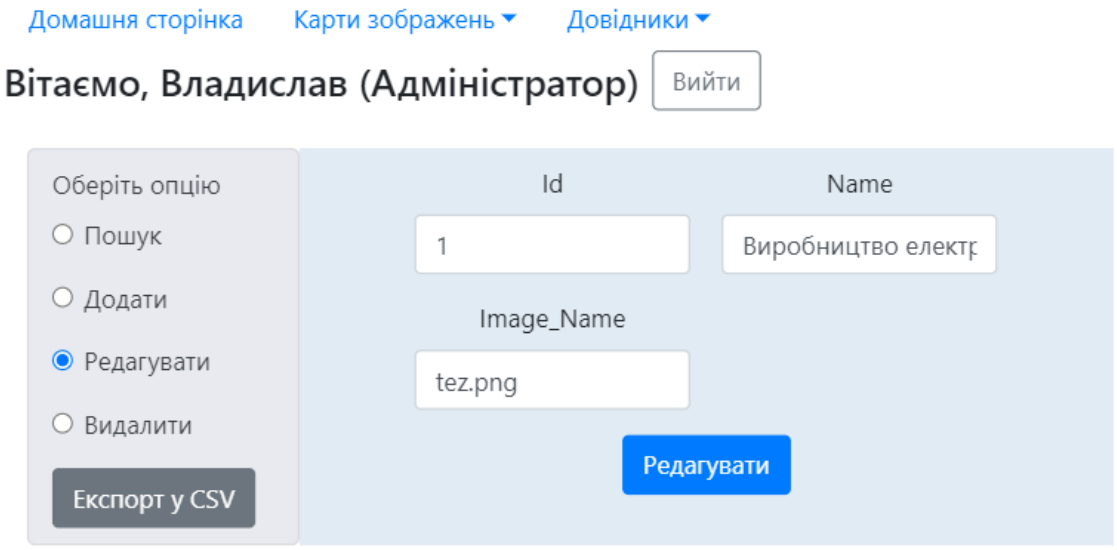


Рисунок 5.24 — Приклад роботи із довідниками із мобільного пристрою.

Для прикладів демонстрації роботи системи на мобільних пристроях було використано Apple IPhone 6.

5.1 Висновки до розділу

У результаті виконання дипломної роботи було створено веб-систему, що дозволяє вносити екологічні показники. Для цього використовується карта, на яку наносяться маркери та полігони з викидами речовин. Система має зручний інтерфейс та дозволяє швидко ввести показники. При роботі з довідниками, користувач завжди бачить актуальний стан даних і будь-які його зміни відразу застосовуються для усієї системи.

Додатково була виконана робота щодо адаптації системи для мобільних пристроїв. Таким чином, фахівець може вносити показники на місці, використовуючи смартфон або планшет.

ВИСНОВКИ

У процесі виконання даної роботи було вивчено предметну область проекту та спроектовано архітектуру клієнт-серверного застосунку, що дозволяє користувачам отримати доступ до системи без зусиль, використовуючи тільки мережу Інтернет. Було досліджено готові реалізації схожих систем, визначено їх основні переваги та недоліки та на основі цих даних був складений список основних вимог до системи.

Виходячи із вимог, було проведено аналіз наявних інструментів та технологій, що дозволяють забезпечити швидкодію та комфорт роботи.

У рамках веб-системи було реалізовано:

1) функціонал для нанесення маркерів на карту;
2) функціонал для нанесення полігонів на карту;
3) функціонал редагування об’єктів;
4) функціонал внесення інформації про викиди залежно від обраного середовища;
5) функціонал, що забезпечує можливість адміністратора додавати нову інформацію у довідники;
6) функціонал, що забезпечує пошук інформації у довідниках;
7) функціонал, що забезпечує можливість адміністратора редагувати інформацію у довідниках;
8) функціонал, що забезпечує можливість адміністратора видаляти інформацію із довідників;

Основна робота із даною веб-системою являє собою взаємодію із картою, що дозволяє зручно задавати параметри для різних об’єктів та бачити загальну картину щодо екологічного стану. Додатково було створено функціонал для роботи з довідниками, із врахуванням прав доступу до функцій редагування і зміни самого довідника та можливістю експорту даних.

Особливу увагу було приділено забезпеченню працездатності системи на мобільних пристроях, оскільки веб-система потенційно може використовуватись для

роботи зі смартфона або планшета.

Таким чином, розроблена веб-система може бути використана фахівцями з галузей екологічного та економічного моніторингу, підприємств чи фірм, що пов’язані із забезпеченням високого рівня життя та належного стану довкілля.

ПЕРЕЛІК ПОСИЛАНЬ

1. Величко О. М. Екологічний моніторинг: Навч. посібник / О. М. Величко, Д.В. Зеркалов.. — К. : Наук. Світ, 2001. — 205 с.

2. Красовский Г. Я. Введение в методы космического мониторинга окружающей среды / Г. Я. Красовский. — Харьков: ХАИ, 1999. — 206 с.

3. Офіційний сайт ЕПК РОСА [Електронний ресурс] — Режим доступу: http://ecolida.ru/zagruzit.

4. Интегральный расчётный код (программный комплекс) Нострадамус [Електронний ресурс] — Режим доступу: http://ibrae.ac.ru/contents/950/.

5. Чучуева И. Модели прогнозирования: общая классификация [Електронний ресурс] / И. Чучуева. — 2012. — Режим доступу: https://www.mbureau.ru/blog/modeli-prognozirovaniya-obshchaya-klassifikaciya.

6. Офіційний сайт RECASS NT [Електронний ресурс] — Режим доступу: http://www.rpatyphoon.ru/products/software-hardware/recass.php

7. Справочник JavaScript [Електронний ресурс] — Режим доступу: https://javascript.info/

8. Пауэлл Т.А. Полное руководство по HTML / Т.А. Пауэлл. — М.: Мир, 2001. — 912 с.

9. Справочник CSS [Електронний ресурс] — Режим доступу: http://htmlbook.ru/css.

10. Офіційний сайт React-Bootstrap [Електронний ресурс] — Режим доступу: https://react-bootstrap.github.io/.

11. Офіційний сайт React [Електронний ресурс] — Режим доступу: https://reactjs.org/.

12. Портнякин И. Эффективные пользовательские интерфейсы / И. Портнякин. — Санкт-Петербург: Лори, 2011. — 600 с.

13. Офіційний сайт Express [Електронний ресурс] — Режим доступу: https://expressjs.com/.

14. Документація MySql [Електронний ресурс] — Режим доступу: http://www.mysql.ru/docs/

15. Документація Ag-Grid [Електронний ресурс] — Режим доступу: https://www.ag-grid.com/

ДОДАТОК 1

Веб-орієнтована система для внесення екологічних параметрів, отриманих з точок збору даних

Специфікація

УКР.НТУУ”КПІ”ім.І.Сікорського. ТР-61121_20Б 81-1

Аркушів 4

2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ“КПІ”.ТР-61 121_20Б 81-1	Записка.docx	Пояснювальна записка
Комплекс		
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-1	Модуль авторизації і автентифікації користувача	
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-2	Модуль HTTP	
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-3	Підсистема вибору та зберігання вибраного середовища	Підсистема для збереження обраного середовища. Вказує середовище для якого будуть задаватись викиди речовин
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-4	Підсистема для відображення карти і об’єктів на ній	
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-5	Підсистема для нанесення/редагування маркерів на карті	Підсистема для нанесення та редагування маркерів на карті, з можливістю додавання викидів речовин та механізмом синхронізації
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-6	Підсистема для нанесення/редагування полігонів на карті	Підсистема для нанесення та редагування полігонів на карті, з можливістю додавання викидів речовин та механізмом синхронізації
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-7	Підсистема побудови графіків викидів	
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-8	Підсистема для роботи із довідником	Підсистема для роботи користувачів та адміністратора із довідником. Містить контроль доступу до операцій пошуку,

		додавання, редагування та видалення.
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-9	Підсистема фільтрації об’єктів на карті	
Компоненти		
УКР.НТУУ“КПІ”.ТР-61 121_20Б 13-1	Програмний_модуль.docx	Опис програмного модулю
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-10	addDictionaryRecord.jsx	Компонент додавання запису у довідник
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-11	addPointModal.jsx	Компонент додавання маркера на карту
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-12	addPolygonModal.jsx	Компонент додавання полігона на карту
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-13	dictionary.jsx	Компонент довідника
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-14	dictionaryModes.jsx	Компонент вибору опцій для роботи із довідником
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-15	editDictionaryRecord.jsx	Компонент редагування запису у довіднику
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-16	removeDictionaryRecord.jsx	Компонент видалення запису у довіднику
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-17	submitForm.jsx	Компонент додавання викиду
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-17	environmentsInfoContext.jsx	Контекст для зберігання інформації про обране середовище
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-18	map.jsx	Компонент карти
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-19	elements.js	Контролер для роботи із елементами
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-20	ownerTypes.js	Контролер для роботи із типами власності
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-21	point.js	Контролер для роботи із маркерами
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-22	polygons.js	Контролер для роботи із

		полігонами
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-23	taxValues.js	Контролер для роботи із податками
УКР.НТУУ“КПІ”.ТР-61 121_20Б 12-24	typeofobjects.js	Контролер для роботи із типами об’єктів

ДОДАТОК 2

Веб-орієнтована система для внесення екологічних параметрів, отриманих з точок збору даних

Лістинг програми

УКР.НТУУ”КПІ”ім.І.Сікорського. ТР-61121_20Б 12-1

Аркушів 11

2020

addPointModal.jsx (Фрагменти)

```
export const AddPointModal = (props) => {
  // ...
  // функція додавання нового маркеру
  const addPoint = (emission) => {
    post(POINT_URL, {
      Name_object: name,
      description,
      type: type.id,
      coordinates,
      emission,
      id_of_user: user.id_of_user,
      owner_type_id: ownerType.id,
    })
      .then(() => {
        clearForm();
        onHide();
        setShouldFetchData(true);
      })
      .catch(() => setShouldFetchData(false));
  };
  // функція редагування маркеру
  const editPoint = (emission) => {
    put(`${POINT_URL}/${pointId}`, {
      Name_object: name,
      description,
      type: type.id,
      owner_type_id: ownerType.id,
      emission,
    })
      .then(() => {
        clearForm();
        onHide();
        setShouldFetchData(true);
        setIsEditPointMode(false);
        setPointId(null);
      })
      .catch(() => {
        setShouldFetchData(false);
        setIsEditPointMode(false);
        setPointId(null);
        setIsEditPointMode(false);
        setPointId(null);
      });
  };
  // функція для підтягування наявних типів об’єктів та форм власності
  useEffect(() => {
    get(TYPE_OF_OBJECT_URL).then(({ data }) => {
      const mappedTypes = data.map(({ Id, Name, Image_Name }) => ({
        id: Id,
        name: Name,
        imageName: Image_Name,
      }));

      setTypes(mappedTypes);
    });
    get(OWNER_TYPES_URL).then(({ data }) => {
      setOwnerTypes(data);
    });
```

```
}, []);
// функція для підтягування інформації про наявний маркер
useEffect(() => {
  if (isEditPointMode && pointId) {
    get(`${POINT_URL}/${pointId}`).then(({ data }) => {
      const type = types.find(({ id }) => id === data.type);
      const ownerType = ownerTypes.find(
        ({ id }) => id === data.owner_type.id
      );
      if (type) {
        setType(type);
      }
      if (ownerType) {
        setOwnerType(ownerType);
      }
      setName(data.Name_object);
      setDescription(data.description);
    });
  }
}, [pointId, isEditPointMode]);
// функція для обробки імпортованого файлу
const fileUpload = async (e) => {
  e.preventDefault();
  if (e.target.files && e.target.files.length) {
    try {
      const type = getUploadedFileType(e.target.files[0]);
      if (type === uploadedFileTypes.txt) {
        const reader = new FileReader();
        reader.onload = async (e) => {
          const mappedResult = await preparedDataPromise(e.target.result);
          setModalFields(mappedResult);
        };
        reader.readAsText(e.target.files[0], 'UTF-8');
      } else if (type === uploadedFileTypes.xlsx) {
        const data = await readXlsxFile(e.target.files[0]);
        setModalFields(data);
      }
    } catch (error) {
      alert('Помилка при обробці вхідних даних');
    }
  }
};
// функція задання полів з імпортованого файлу у форму
const setModalFields = (rows) => {
  let preloadedEmission = null;
  const actionsMap = new Map([
    [
      'OBJECT TYPE',
      (columnValue) => {
        const type = types.find(({ name }) => name === columnValue);
        setType(type);
      },
    ],
    [
      'OWNER TYPE',
      (columnValue) => {
        const type = ownerTypes.find(({ type }) => type === columnValue);
        setOwnerType(type);
      },
```

```
      ],
      ['NAME', (columnValue) => setName(columnValue)],
      ['DESCRIPTION', (columnValue) => setDescription(columnValue)],
      [
        'DATE',
        (columnValue) =>
          (preloadedEmission = { ...preloadedEmission, date: columnValue }),
      ],
      [
        'ELEMENT',
        (columnValue) =>
          (preloadedEmission = {
            ...preloadedEmission,
            elementName: columnValue,
          }),
      ],
      [
        'AVERAGE VALUE',
        (columnValue) =>
          (preloadedEmission = {
            ...preloadedEmission,
            averageValue: columnValue,
          }),
      ],
      [
        'MAXIMUM VALUE',
        (columnValue) =>
          (preloadedEmission = {
            ...preloadedEmission,
            maximumValue: columnValue,
          }),
      ],
    ]);
    try {
      rows.forEach(([columnName, columnValue]) =>
        actionsMap.get(columnName)(columnValue)
      );
      setPreloadedEmission(preloadedEmission);
    } catch (error) {
      alert('Помилка. Неправильні дані');
      console.error(error);
    }
  };
  // ...
```

addPolygonModal.jsx (Фрагменти)

```
export const AddPolygonModal = (props) => {
  // ...
  // функція для підтягування інформації про наявний полігон
  useEffect(() => {
    if (polygonId && isEditPolygonMode) {
      get(`${POLYGON_URL}/${polygonId}`).then(({ data }) => {
        setLineThickness(data.line_thickness);
        setColor({
          r: data.brush_color_r,
          g: data.bruch_color_g,
          b: data.brush_color_b,
          a: data.brush_alfa,
        });
        setName(data.name);
```

```
        setDescription(data.description);
      });
    }
  }, [polygonId, isEditPolygonMode]);
  // функція додавання полігону
  const addPolygon = (emission) => {
    post(POLYGON_URL, {
      brush_color_r: color.r,
      bruch_color_g: color.g,
      brush_color_b: color.b,
      brush_alfa: color.a,
      line_collor_r: color.r,
      line_color_g: color.g,
      line_color_b: color.b,
      line_alfa: color.a,
      line_thickness: Number(lineThickness),
      name,
      id_of_user: Number(user.id_of_user),
      type: initialState.form.type,
      description,
      points: coordinates.map((point, index) => ({
        latitude: point.lat,
        longitude: point.lng,
        order123: index + 1,
      })),
      emission,
    })
      .then(() => {
        clearForm();
        onHide();
        setNewPolygonCoordinates([]);
        setShouldFetchData(true);
      })
      .catch(() => {
        setNewPolygonCoordinates([]);
        setShouldFetchData(false);
      });
  };
  // функція редагування полігону
  const editPolygon = (emission) => {
    put(`${POLYGON_URL}/${polygonId}`, {
      brush_color_r: color.r,
      bruch_color_g: color.g,
      brush_color_b: color.b,
      brush_alfa: color.a,
      line_collor_r: color.r,
      line_color_g: color.g,
      line_color_b: color.b,
      line_alfa: color.a,
      line_thickness: Number(lineThickness),
      name,
      description,
      emission,
    })
      .then(() => {
        clearForm();
        onHide();
        setNewPolygonCoordinates([]);
        setShouldFetchData(true);
```

```
        setIsEditPolygonMode(false);
        setPolygonId(null);
      })
      .catch(() => {
        setIsEditPolygonMode(false);
        setPolygonId(null);
        setNewPolygonCoordinates([]);
        setShouldFetchData(false);
      });
  };
// ...
```

submitForm.jsx (Основні фрагменти)

```
export const SubmitForm = ({ onSave, preloadedEmission }) => {
  const { environmentsInfo } = useContext(EnvironmentsInfoContext);
  // ...
  // підтвердження вводу вхідних параметрів у форму
  const handleSubmit = () => {
    let emission;
    if (isActive && date) {
      const [year, month, day] = date.split('-');
      emission = isActive && {
        valueAvg,
        valueMax,
        year,
        month,
        day,
        idElement: selectedElement.code,
        idEnvironment: environmentsInfo.selected.id,
        measure,
      };
    }
    onSave(emission);
    clearForm();
  };
  // функція синхронізації введених показників із наявними у базі даних
  const selectElement = (element) => {
    setElement(element);
    setMeasure(element.measure);
    post(GDK_FIND_URL, {
      code: element.code,
      environment: environmentsInfo.selected.id,
    }).then(({ data }) => {
      if (data.average && data.max) {
        setGdkAvg(data.average);
        setGdkMax(data.max);
      } else {
        setGdkAvg(initialState.form.valueAvg);
        setGdkMax(initialState.form.valueMax);
      }
    });
  };
// ...
```

elements.js (Фрагменти)

```
// функція вибору всіх елементів із бази даних
const getElements = async (req, res) => {
  const getElementsPromise = new Promise((resolve, reject) => {
    const query = `SELECT * FROM ??;`;
    const values = [tableName];
    return pool.query(query, values, (error, rows) => {
```

```
      if (error) {
        return reject(error);
      }

      return resolve(rows);
    });
  });
  try {
    const rows = await getElementsPromise;
    return res.send(JSON.stringify(rows));
  } catch (error) {
    return res.status(500).send({ message: error });
  }
};
// функція додавання нового елемента у базу даних
const addElement = async (req, res) => {
  const addElementPromise = new Promise((resolve, reject) => {
    const query = `INSERT INTO ?? VALUES(?)`;
    pool.query(query, [tableName, Object.values(req.body)], (error, rows) => {
      if (error) {
        return reject(error);
      }
      if (rows.affectedRows === 1) {
        return resolve();
      }
    });
  });
  try {
    await addElementPromise;
    return res.sendStatus(200);
  } catch (error) {
    return res.status(500).send({ message: error });
  }
};
// функція оновлення наявного у базі даних елемента
const editElement = async (req, res) => {
  const editElementPromise = new Promise((resolve, reject) => {
    const id = req.params.id;
    const { body: updatedValues } = req;
    const query = `UPDATE ?? SET ? WHERE ?? = ?`;
    const values = [tableName, updatedValues, 'code', id];
    pool.query(query, values, (error, rows) => {
      if (error) {
        return reject(error);
      }
      if (rows.affectedRows === 1) {
        return resolve();
      }
    });
  });
  try {
    await editElementPromise;
    return res.sendStatus(200);
  } catch (error) {
    return res.status(500).send({ message: error });
  }
};
// функція видалення елемента з бази даних
const removeElement = async (req, res) => {
  const removeElementPromise = new Promise((resolve, reject) => {
```

```
      const id = req.params.id;
      const query = `DELETE FROM ?? WHERE ?? = ?`;
      const values = [tableName, 'code', id];
      pool.query(query, values, (error, rows) => {
        if (error) {
          return reject(error);
        }
        if (rows.affectedRows === 1) {
          return resolve();
        }
      });
    });
    try {
      await removeElementPromise;
      return res.sendStatus(200);
    } catch (error) {
      return res.status(500).send({ message: error });
    }
  };
  // …
```

point.js (Фрагменти)

```
  // …
  // функція додавання точки у базу даних
  const addPoint = (req, res) => {
    const {   Name_object,   type,   coordinates,   description,   emission,   id_of_user,
owner_type_id: owner_type, } = req.body;
    const query = `
    INSERT INTO poi
    (id_of_user, Type, owner_type, Coord_Lat, Coord_Lng, Description, Name_object)
    VALUES ('${id_of_user}', '${type}', '${owner_type}','${coordinates[0]}', '${coordinates[1
]}', '${description}', '${Name_object}');`;
    const pointPromise = new Promise((resolve, reject) => {
      pool.query(query, (error, rows) => {
        if (error) {
          reject(error);
        }
        resolve(+rows.insertId);
      });
    });
    return pointPromise
      .then((insertedId) => {
        if (!!emission) {
          emission.idPoi = insertedId;
          return insertEmissionOnMap(SOURCE_POI, emission);
        }
      })
      .then(() => res.sendStatus(200))
      .catch((error) => {
        res.status(500).send({
          message: error,
        });
      });
  };
  // функція отримання інформації про конкретну точку
  const getPoint = (req, res) => {
    const id = req.params.id;
    const poiPromise = new Promise((resolve, reject) => {
      const query = `SELECT id_of_user, poi.type,
description,  Name_object,  owner_type as owner_type_id,
```

```
        owner_types.type as owner_type_name
      FROM poi
      INNER JOIN owner_types ON poi.owner_type = owner_types.id
      WHERE poi.Id = ${id};`;
      pool.query(query, [], (error, rows) => {
        if (error) {
          reject(error);
        }
        rows = rows.map(
          ({ id_of_user, type, description, Name_object, owner_type_id, owner_type_name }) =>
{
            return {
              id_of_user, type, description, Name_object,
              owner_type: { id: owner_type_id, name: owner_type_name, },
            };
          }
        );
        if (rows[0]) {
          resolve(rows[0]);
        }
      });
    });
    return poiPromise
      .then((poi) => res.send(poi))
      .catch((error) => res.status(500).send({ message: error }));
  };
  // функція оновлення наявної точки
  const updatePoint = (req, res) => {
    const id = req.params.id;
    const { Name_object, description, type, emission, owner_type_id: owner_type } = req.body;
    const poiPromise = new Promise((resolve, reject) => {
      const tableName = 'poi';
      const updatedValues = { Name_object, description, type, owner_type, };
      const query = `UPDATE ?? SET ? WHERE ?? = ?`;
      const values = [tableName, updatedValues, 'Id', id];
      pool.query(query, values, (error, rows) => {
        if (error) {
          reject(error);
        }
        if (rows) {
          resolve();
        }
      });
    });
    return poiPromise
      .then(() => {
        if (!!emission) {
          emission.idPoi = +id;
          return insertEmissionOnMap(SOURCE_POI, emission);
        }
      })
      .then(() => res.sendStatus(200))
      .catch((error) => res.status(500).send({ message: error }));
  };
```

polygons.js (Фрагменти)

```
  // ...
  // функція додавання полігону у базу даних
  const addPolygon = (req, res) => {
    const lastIdPromise = new Promise((resolve, reject) => {
```

```
    const lastIdQuery = `SELECT MAX(??) as maxId FROM ??;`;
    pool.query(lastIdQuery, ['id_of_poligon', 'poligon'], (error, rows) => {
      if (error) {
        reject(error);
      }
      resolve(rows[0].maxId);
    });
  });
  lastIdPromise
    .then((maxId) => {
      const id = maxId + 1;
      const { points, emission, ...values } = req.body;
      const insertPolygonPromise = new Promise((resolve, reject) => {
        const insertPolygonQuery = `INSERT INTO ?? VALUES (?)`;
        pool.query(
          insertPolygonQuery,
          ['poligon', [id, ...Object.values(values)]],
          (error) => {
            if (error) {
              reject(error);
            }
            resolve(id);
          }
        );
      });
      return insertPolygonPromise.then((id) => id);
    })
    .then((id) => {
      const { emission } = req.body;
      if (!!emission) {
        emission.idPolygon = id;
        return insertEmissionOnMap(SOURCE_POLYGON, emission).then(() => id);
      }
      return id;
    })
    .then((id) => {
      const { points } = req.body;
      const insertPolygonPointsPromises = points.map(
        ({ latitude, longitude, order123 }) => {
          return new Promise((resolve, reject) => {
            const insertPolygonPointsQuery = `INSERT INTO ?? VALUES (?)`;
            pool.query(
              insertPolygonPointsQuery,
              ['point_poligon', [latitude, longitude, id, order123]],
              (error) => {
                if (error) {
                  reject(error);
                }
                resolve();
              }
            );
          });
        }
      );
      return Promise.all(insertPolygonPointsPromises);
    })
    .then(() =>  res.sendStatus(200))
    .catch((error) => res.status(500).send({ message: error, }));
};
```

```
    // функція отримання інформації про конкретний полігон
    const getPolygon = (req, res) => {
      const id = req.params.id;
      const polygonPromise = new Promise((resolve, reject) => {
        const tableName = 'poligon';
        const query = `SELECT ?? FROM ?? WHERE ?? = ?`;
        const values = [
          [
            'brush_color_r', 'bruch_color_g',  'brush_color_b',  'brush_alfa', 'line_collor_r',
'line_color_g', 'line_color_b', 'line_alfa', 'line_thickness', 'name', 'id_of_user', 'type',
'description',], tableName, 'Id_of_poligon', id,];
        pool.query(query, values, (error, rows) => {
          if (error) {
            reject(error);
          }
          if (rows[0]) {
            resolve(rows[0]);
          }
        });
      });
      return polygonPromise
        .then((polygon) => res.send(polygon))
        .catch((error) => res.status(500).send({ message: error }));
    };
    // функція оновлення наявного у базі даних полігону
    const updatePolygon = (req, res) => {
      const id = req.params.id;
      const {   brush_color_r,   bruch_color_g,   brush_color_b,   brush_alfa,   line_collor_r,
line_color_g,   line_color_b,   line_alfa,   line_thickness,   name,   description,   emission,
} = req.body;
      const polygonPromise = new Promise((resolve, reject) => {
        const tableName = 'poligon';
        // ...
        const query = `UPDATE ?? SET ? WHERE ?? = ? `;
        const values = [tableName, updatedValues, 'Id_of_poligon', id];
        pool.query(query, values, (error, rows) => {
          if (error) {
            reject(error);
          }
          if (rows) {
            resolve();
          }
        });
      });
      return polygonPromise
        .then(() => {
          if (!!emission) {
            emission.idPolygon = +id;
            return insertEmissionOnMap(SOURCE_POLYGON, emission);
          }
        })
        .then(() => res.sendStatus(200))
        .catch((error) => res.status(500).send({ message: error }));
    };
```

ДОДАТОК 3

Веб-орієнтована система для внесення екологічних параметрів, отриманих з точок збору даних

Програмний модуль

УКР.НТУУ”КПІ”ім.І.Сікорського. ТР-61121_20Б 13-1

Аркушів 13

2020

АНОТАЦІЯ

Розроблені підсистеми для введення даних є окремими складовими всієї веб-системи. Вони можуть працювати як самостійні блоки, взаємодіючи через визначене API.

Функціонал підсистем, розроблених у роботі, включає:

— Нанесення маркерів на карту способами ручного вводу або імпорту Excel чи текстового файлу;

— Нанесення полігонів на карту способами ручного вводу або імпорту Excel або текстового файлу;

— Внесення викидів речовин для полігонів та маркерів;

— Миттєве порівняння введених показників із гранично допустимими значеннями;

— Додавання нової інформації у довідники, використовуючи графічний інтерфейс;

— Редагування наявної інформації у довідниках, використовуючи графічний інтерфейс;

— Видалення інформації з довідників, використовуючи графічний інтерфейс.

Засоби розробки: середовища розробки WebStorm IDEA, Visual Studio Code, мова програмування JavaScript, платформа NodeJS, система контролю версій Git, СКБД MySQL.

ЗМІСТ

1. ЗАГАЛЬНІ ВІДОМОСТІ

Мова програмної реалізації — JavaScript. Середовища розробки — WebStorm IDEA 2020.1.1, Visual Studio Code 1.45.1.

Для роботи із програмним забезпеченням потрібно мати:

— Пристрій, де відбуватиметься робота;

— Доступ до мережі Інтернет;

— Браузер.

2. ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Підсистеми для роботи із маркерами, полігонами та довідниками були створені з метою забезпечення комфортної можливості оновлення бази знань щодо екологічного стану середовища. На рисунку 2.1 вони виділені синім кольором.

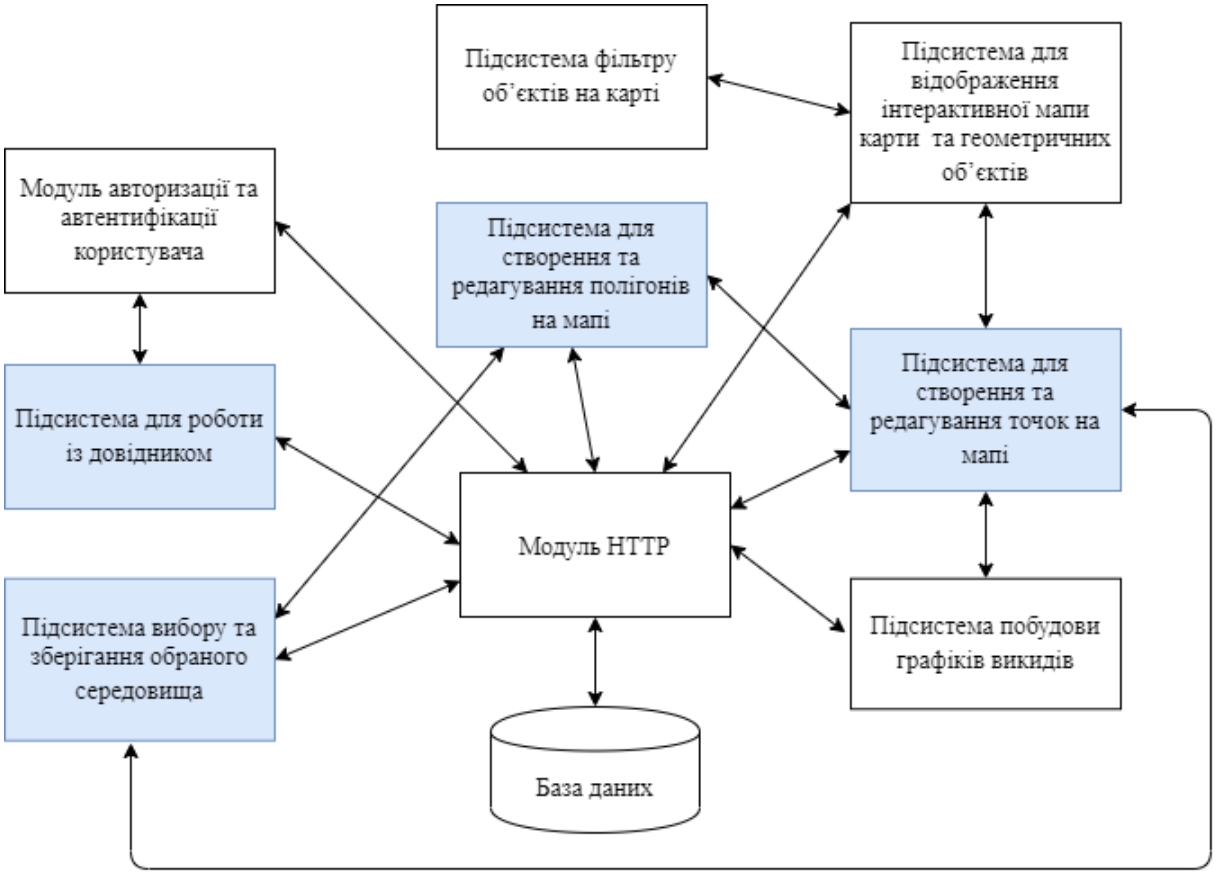


Рисунок 2.1 — Підсистеми для роботи з маркерами, полігонами та довідниками

Експерт, використовуючи ці модулі, має змогу вносити показники, не використовуючи спеціальної апаратури та спеціального програмного забезпечення. Експерт може:

— нанести на карту маркер і додати інформацію про викиди речовин;

— нанести на карту цілу область і додати інформацію про викиди речовин у

ній;

— додати викиди речовин до наявних полігонів та маркерів;

— отримати аналіз щодо вхідних параметрів ще на етапі введення;

— шукати інформацію по довідниках.

Додатково адміністратор може:

— додати нову інформацію у довідник;

— редагувати наявну інформацію у довіднику;

— видалити інформацію з довідника.

3. ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Логіка роботи системи спрямована на комунікацію між клієнтською частиною та серверною. Алгоритм роботи полягає в обміні інформацією, використовуючи можливості різних типів HTTP-запитів.

Дії користувача, при внесенні або зміні параметрів, ініціюють запит на сервер, що його обробляє. Оскільки система є веб-орієнтованою, всі дії відбуваються у мережі Інтернет по визначених URL-шляхах. Тому у системі визначені шляхи для клієнтської частини та API шляхи.

Робота сервера полягає у тому, щоб зрозуміти, що за запит прийшов і як його потрібно обробляти. Після того як сервер визначив запит, інформація запиту відправляється у відповідний обробник контролера, де отримана інформація підготовлюється для формування SQL-запитів до сервера бази даних. Після цього, вони відправляються до бази даних і розглядається результат операції. У випадку помилки, сервер відсилає повідомлення про помилку на клієнтську частину, якщо усе пройшло успішно, сервер обробляє отримані дані та підготувавши їх, відправляє назад до клієнтської сторони зі статусом 200.

Робота клієнтської частини полягає у забезпеченні комфортного графічного інтерфейсу для користувача та обробку його дій. При зміні параметрів, система реактивно на це реагує і показує актуальний стан, що забезпечує передбачувану для користувача поведінку.

4. ВИКЛИК І ЗАВАНТАЖЕННЯ

Для того, щоб почати користуватись системою, необхідно зайти на сайт веб-системи, використовуючи браузер комп’ютера або мобільного пристрою. Для з’єднання необхідно мати доступ до мережі Інтернет.

5. ВХІДНІ ДАНІ

Оскільки веб-система дозволяє наносити на карту об’єкти різних типів, вхідні дані мають дещо різну структуру.

Для нанесення маркеру необхідно, у режимі додавання маркера, обрати на карті точку, де він буде розміщений. Після цього ввести наступні дані:

— обрати зі списку тип об’єкту, що додатково буде задавати іконку маркера;

— обрати форму власності об’єкту;

— ввести ім’я маркера;

— ввести опис маркера;

— додатково можна ввести інформацію про викид на цьому об’єкті.

Для нанесення полігону необхідно, у режимі додавання полігону, послідовно вказати точки на карті, які будуть формувати область й натиснути кнопку завершення виділення полігону. Після цього внести інформацію про цей полігон. Для нього задаються наступні параметри:

— товщина лінії обводу;

— колір полігону;

— ім’я полігону;

— опис полігону;

— додатково можна ввести інформацію про викид на цій області.

При додаванні викиду користувач повинен:

— обрати із календаря дату викиду;

— ввести середнє значення викиду;

— ввести максимальне значення викиду;

— обрати елемент з списку.

Ці значення порівнюються з наявними гранично допустимими, тому, у процесі введення, експерт може відразу робити певний аналіз отриманих даних.

Важливо зазначити, що користувач має можливість не вводити всі дані вручну, а імпортувати дані з Excel чи текстового файлу. При цьому є валідація на правильність

формату даних в імпортованому файлі. На рисунку 5.1 та 5.2 зображені приклади таких даних для маркера й полігону відповідно.

	A	B
1	OBJECT_TYPE	Забір, очищення та постачання води
2	OWNER_TYPE	Комунальна
3	NAME	Київводоканал
4	DESCRIPTION	Київський водоканал
5	DATE	28.12.2015
6	ELEMENT	Барий оксид
7	AVERAGE_VALUE	4
8	MAXIMUM_VALUE	3

Рисунок 5.1 — Формат даних імпорту для маркера

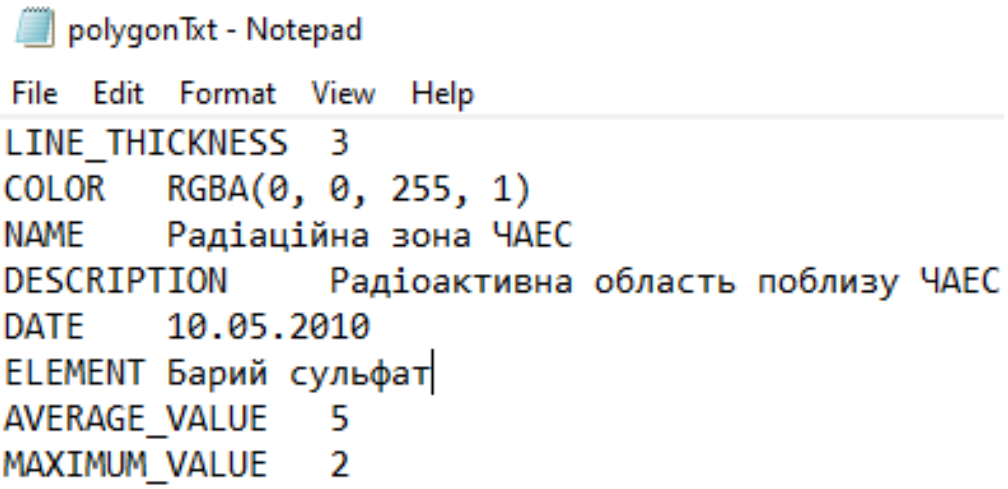


Рисунок 5.2 — Формат даних імпорту для полігону

При роботі із довідником система підтягує дані у залежно від обраного довідника. На сторінці довідника користувач обирає із доступних опцій потрібну. Після цього, включається відповідний режим роботи із довідником. При додаванні даних показуються колонки з полями для введення з обраної таблиці. Редагування відбувається аналогічним чином, з відмінністю того, що поля автоматично

заповнюються у залежності від вибраного запису із довідника. Пошук відбувається по усім можливим збігам. Для видалення необхідно обрати елемент і додатково підтвердити операцію. Вхідними даними для довідника є:

— для додавання й редагування — нові значення полів, що будуть заноситись у відповідні колонки довідника (рисунок 5.3);

— для пошуку — пошуковий запит;

— для видалення — ідентифікатор вибраного обраного запису;

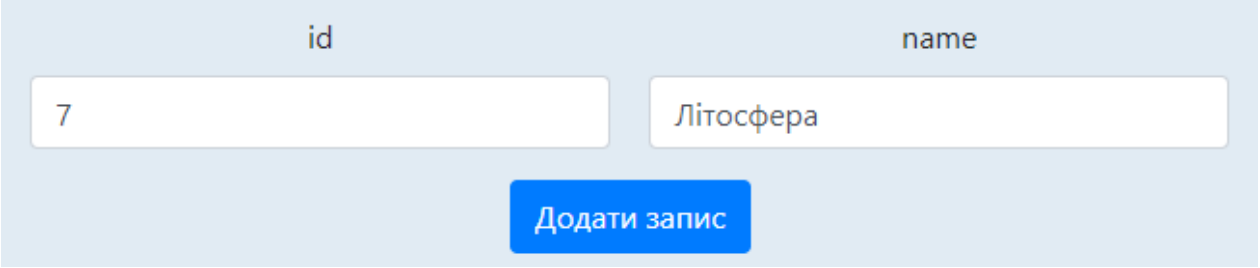


Рисунок 5.3 — Додавання нового запису у довідник

6. ВИХІДНІ ДАНІ

Вихідними даними є відповідні записи у таблицях бази даних. Після додавання маркера чи полігону на карту, у таблиці бази даних створюється новий запис, що сприяє показу нового об’єкту на карті (рисунки 6.1, 6.2, 6.3 та 6.4).

810	4	4	5	62.75...	7.119...	Київськи...	Київводоканал	0	0	0

Рисунок 6.1 — Запис нового маркера у базі даних

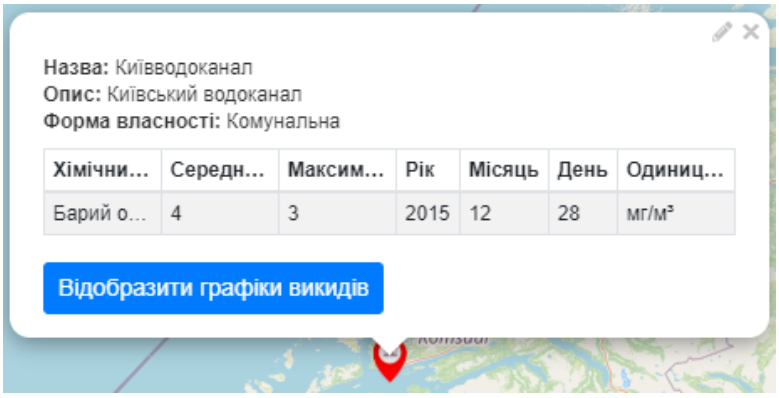


Рисунок 6.2 — Показ нового маркера на карті

39	0	0	255	1	0	0	255	1	3	Радіаційна зона ЧАЕС	4	poligon	Радіоактивна область поблизу ЧАЕС

Рисунок 6.3 — Запис нового полігону у базі даних

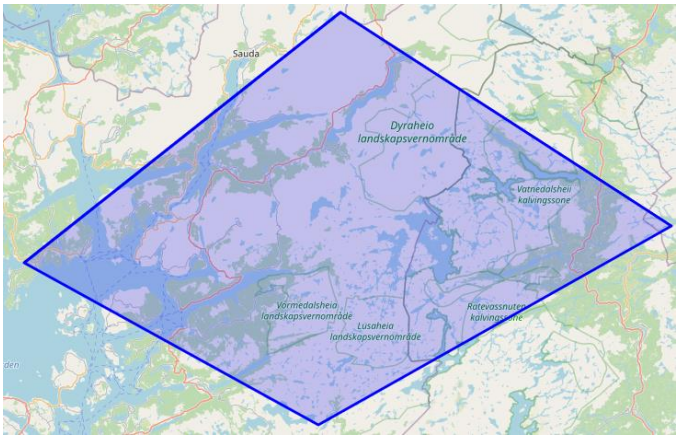
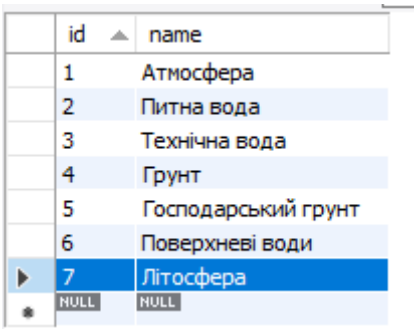


Рисунок 6.4 — Показ нового полігону на карті

Зміни в довідниках зберігаються у віддаленій базі даних і відразу застосовуються для наявних об’єктів. Отже, коли адміністратор додає нову або змінює наявну інформацію у довідниках, система відразу застосовує нові зміни та завжди знаходиться в актуальному стані.



id	name
1	Атмосфера
2	Питна вода
3	Технічна вода
4	Грунт
5	Господарський грунт
6	Поверхневі води
7	Літосфера
NULL	NULL

Рисунок 6.4 — Новий запис, який було додано із довідника